

DTIC FILE COPY

December 1989

UILU-ENG-89-2243
DAC-16

2

COORDINATED SCIENCE LABORATORY
College of Engineering

AD-A217 459

DTIC
ELECTE
JAN 29 1990
S D D

**iSITE:
AUTOMATIC
CIRCUIT SYNTHESIS
FOR DOUBLE-METAL
CMOS VLSI
CIRCUITS**

Perry Gee

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Approved for Public Release. Distribution Unlimited.

90 01 26 0 15

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified			1b. RESTRICTIVE MARKINGS None	
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE				
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-89-2243 (DAC-16)			5. MONITORING ORGANIZATION REPORT NUMBER(S)	
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois		6b. OFFICE SYMBOL (if applicable) N/A	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research SRC and TI	
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Ave. Urbana, IL 61801			7b. ADDRESS (City, State, and ZIP Code) Arlington, VA 22217 (JSEP) Research Triangle Park, NC 27709 (SRC)	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Joint Services Electronics Program & SRC & TI		8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-84-C-0149 SRC 88-DP-109	
8c. ADDRESS (City, State, and ZIP Code) Arlington, VA 22217 Research Triangle Park, NC 27709			10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. PROJECT NO. TASK NO. WORK UNIT ACCESSION NO.	
11. TITLE (Include Security Classification) "iSITE: Automatic Circuit Synthesis for Double-Metal CMOS VLSI Circuits"				
12. PERSONAL AUTHOR(S) Gee, Perry				
13a. TYPE OF REPORT Technical		13b. TIME COVERED FROM 8/83 TO 9/89	14. DATE OF REPORT (Year, Month, Day) 1989 December	15. PAGE COUNT 139
16. SUPPLEMENTARY NOTATION				
17. COSATI CODES FIELD GROUP SUB-GROUP			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Synthesis, cell generation, physical layout, symbolic layout, metal-metal matrix, logic design compaction, delay optimization.	
19. ABSTRACT (Continue on reverse if necessary and identify by block number) Very large scale integrated (VLSI) circuit technology allows one to manufacture chips with several million devices. Designing such large circuits cannot be accomplished without design automation tools and computer-aided design tools. This thesis addresses the problem of automatic circuit synthesis for double-metal CMOS technology. Large circuits are partitioned into cells and represented as incidence matrices. The rows and columns of these matrices are folded to minimize the area. A symbolic layout is then generated for each matrix. This symbolic layout is then used to generate the physical mask layers necessary for fabrication in the metal-metal matrix methodology.				
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL			22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL

BY

B.S., University of California, 1983
M.S., University of Illinois, 1985



Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1989

Urbana, Illinois

[illegible]

© Copyright by Perry Gee, 1989

ABSTRACT

✓ Very large scale integrated (VLSI) circuit technology allows one to manufacture chips with several million devices. Designing such large circuits cannot be accomplished without design automation tools and computer-aided design tools. This thesis addresses the problem of automatic circuit synthesis for double-metal CMOS technology. Large circuits are partitioned into cells and represented as incidence matrices. The rows and columns of these matrices are folded to minimize the area. A symbolic layout is then generated for each matrix. This symbolic layout is then used to generate the physical mask layers necessary for fabrication in the metal-metal matrix methodology.

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation and gratitude to my advisor Professor Ibrahim Hajj, and my co-advisor, Professor Sung-Mo (Steve) Kang, for their invaluable guidance throughout the course of this research. I would also like to thank Professors Prithviraj Banerjee and Chung-Laung (Dave) Liu for being members of my dissertation committee. Special thanks to David Overhauser and Jerome Chen, for without their assistance, knowledge, and friendship, this research would not have been possible.

I would like to thank my parents and my family for their encouragement and support. I wish to express my gratitude to Arlene and Robert Causey, for without them, my final year of graduate study would have been unbearable. Gerald and Patti Nedoluha also deserve special recognition for their valuable friendship throughout my undergraduate and graduate studies.

Finally, I would like thank Jim Krogmier, Lee Potter, Andrew Yang, Andy Sustich, Jeremy Grace, Chuck Yeung, Edwardo Acuna, Shun-Lin Su, and the members of the VLSI Circuits Group.

This research was supported in part by a contract from the Joint Services Electronics Program and the Semiconductor Research Corporation, and a grant from Texas Instruments.

TABLE OF CONTENTS

CHAPTER	PAGE
1. INTRODUCTION	1
1.1. Architectural design	3
1.2. Logic synthesis and design	4
1.3. Gate/circuit design	5
1.4. Layout design	5
1.5. Simulation and verification	8
1.6. Overview	8
2. BACKGROUND	13
2.1. Programmable logic arrays (PLAs)	14
2.2. Weinberger array	18
2.3. Gate matrix	18
2.4. Metal-metal matrix (M^3) layout	23
3. SYMBOLIC CIRCUIT REPRESENTATION AND GENERATION	30
3.1. Incidence matrices	30
3.2. Incidence matrix compaction	30
3.2.1. Symbolic matrix compaction methodologies	36
3.2.2. Column merging	38
3.2.3. Row merging	41
3.2.3.1. Min-net-cut across columns algorithm	43
3.2.3.2. CMOS column ordering	48
3.2.3.3. Constraint detection	52
3.2.3.4. Track assignment	57
3.3. Summary	58
4. SYMBOLIC LAYOUT	60
4.1. Introduction	60
4.2. Symbolic layout system overview	61
4.3. Layout symbols and symbol mapping	63
4.4. Grid spacing	67
4.5. Results and comparisons	69
4.6. Summary	73

5. iSITE: AUTOMATIC MODULE GENERATION SYSTEM	75
5.1. System overview	75
5.2. Circuit specification	77
5.3. Circuit grouping and partitioning	79
5.4. Circuit extraction, recognition, and optimization	83
5.5. Results	87
6. CONCLUSIONS AND FUTURE RESEARCH	89
6.1. Conclusions	89
6.2. Future research	91
APPENDIXA.PASS TRANSISTOR DETECTION	94
APPENDIXB.32-BIT CARRY-LOOK-AHEAD ADDER	98
APPENDIXC.AN NP-COMPLETE PROBLEM -- MINIMUM TOTAL 1- 1 DISTANCE	101
APPENDIXD.iSITE USER'S MANUAL	104
D.1. Introduction	104
D.2. Circuit synthesis with iSITE	105
D.2.1. Generating compacted incidence matrices (iSPICE2AA)	106
D.2.2. Prefolding analysis (iPREFOLD)	111
D.2.3. Symbolic layout (iLAYOUT)	111
D.2.4. Physical layout (iSILVER)	115
D.2.5. Gate recognition (iD2GATE)	119
D.2.6. Updating transistor sizes (iUPDTRAN)	121
REFERENCES	123
VITA	130

LIST OF TABLES









TABLE	PAGE
3.1. A Comparison of Improved Min-Net-Cut Algorithm with Conventional Min-Net-Cut for Metal-Metal-Matrix Layouts	47
3.2. A Comparison of CMOS Ordering Algorithm with K-L Min-Net-Cut for Metal-Metal-Matrix Layouts	52
4.1. Layout Symbols	66
4.2. Layout Area for PLA, Gate Matrix, and M^3	69
4.3. Layout Area for Standard Cells and M^3	70
4.4. The Impact of Different Transistor Sizes on Circuit Area	72
5.1. A Comparison of Partitioning Strategies	87

LIST OF FIGURES

FIGURE	PAGE
1.1. The Progression of VLSI Circuit Design	2
1.2. Complex Cell Design	9
2.1. Programmable Logic Array	15
2.2. Conceptual Diagram of Storage Logic Array	17
2.3. Weinberger Array	19
2.4. Gate Matrix	21
2.5. Controlling the Threshold Voltage in Transistors	24
2.6. Metal-Metal Matrix	26
3.1. Circuit Schematic and its Circuit Graph	31
3.2. Incidence Matrix	32
3.3. Column Folding Methodologies	33
3.4. Feature Separation for 2 micron Process	35
3.5. An Example of Min-Cut Ordering for Minimizing the Total 1-1 Distance	40
3.6. Column Compacted Incidence Matrix	41
3.7. Effect of Column Ordering in Metal-Metal Matrix Layouts	42
3.8. Net-Density Between Columns is a Poor Estimate of Track Requirements	43
3.9. Example of Min-Net-Cut Exchanges	44
3.10. Effectiveness of Heuristics	46
3.11. CMOS Ordering Algorithm	49
3.12. CMOS Compacted Incidence Matrix	50
3.13. Net Graph Before and After Column Ordering	51
3.14. Compacted Incidence Matrix and its Constraint Graph	54
3.15. Horizontal Constraints From Net E in Figure 3.14 can be Removed by Adding an Additional Column to the Layout	55
3.16. Effect of Removing the Vertical Constraints from Net E on the Constraint Graph	55
3.17. Vertical Constraint Graph	58
3.18. Symbolic Layout	59
4.1. Technology-File for 2-Micron CMOS Twin-Tub Process	62
4.2. Circuit Schematic, Symbolic Layout, and Mask Data	64
4.3. Progression of Contact Orientation Algorithm	68

4.4.	An M^3 Layout of the 74181 4-Bit ALU	74
5.1.	iSITE System Overview	76
5.2.	iSITE Predefined Library of Gates	78
5.3.	JK FLIP-FLOP	79
5.4.	Channel-Connected Transistor Blocks	81
5.5.	Super-Transistor Decomposition	84
5.6.	XOR and XNOR Pass Transistor Gates	85
5.7.	Channel Connected Transistors	86

LIST OF SYMBOLS

	p-well
	n-diffusion
	p-diffusion
	poly
	metal 1
	contact
	metal 2
	metal-metal via

CHAPTER 1.

INTRODUCTION

The VLSI engineer faces an increasingly difficult problem as the level of integration increases. Today, VLSI chips with several million devices can be manufactured [Koba89a]. In order to cope with the problem of designing such large circuits, engineers have employed a hierarchical design methodology. In addition to this hierarchy, computers have become an integral part of the design process. Computers are used extensively at each stage of the design process to increase the productivity of the engineer. However, as the number of devices on a chip continues to increase, more sophisticated software tools must be developed in order for the design of complex systems to be completed in a reasonable amount of time.

The current approach to designing VLSI systems can be divided into the following tasks, which are shown in Figure 1.1:

- 1) Architectural Design
- 2) Logic Synthesis and Design
- 3) Gate/Circuit Design
- 4) Layout
- 5) Simulation and Verification
- 6) Reiterate the above until specifications are met.

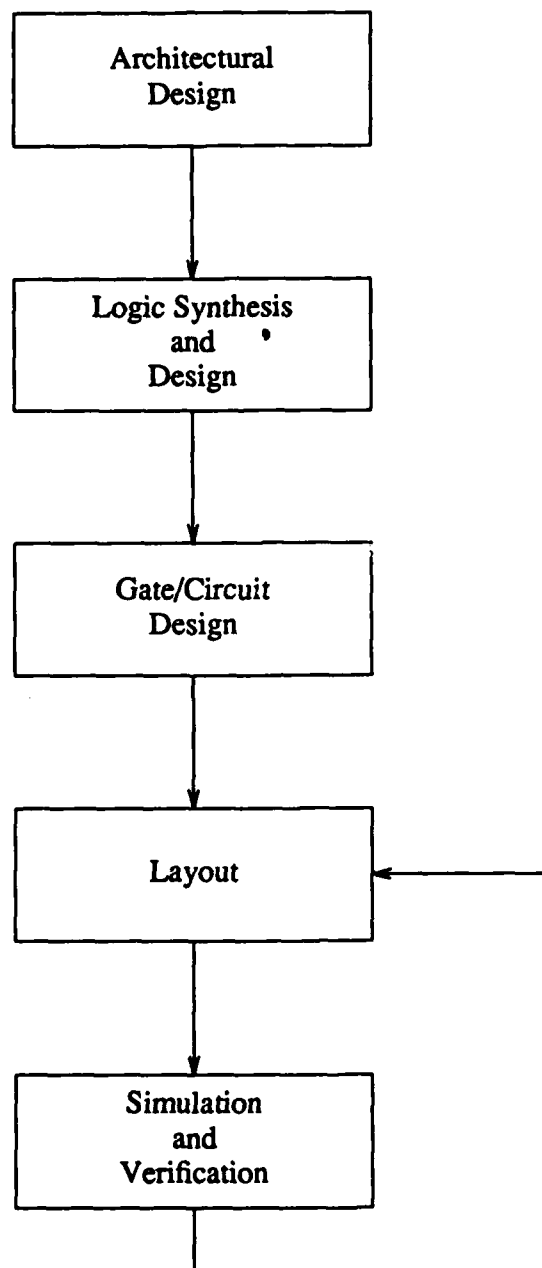


Figure 1.1. The Progression of VLSI Circuit Design

1.1. Architectural design

Architectural design provides the most abstract view of a circuit. The goal of architectural design is to specify the behavioral aspects of the circuit, such as, its functionality and interfacing requirements. The general features that are usually decided on in this phase are the circuit's architecture, such as, a sequential machine design or a pipelined machine design, the number and width of busses, and the placement and size estimates of large functional blocks. This specification can be captured by using flowcharts, or a hardware description language. The hardware description language approach is becoming more popular for several reasons. First, hardware description languages contain constructs for expressing the circuit in a hierarchal and structured fashion. This allows circuits to be written as a collection of modules. Each module performs a well-defined function and can be built up from simpler modules. Each of these modules can be tested independently, which reduces the time necessary to verify the entire system. Also, this hierarchal decomposition of the system aids in the conceptual understanding of the circuit, since modules which use a particular module need to know only about its interfacing requirements and not its implementation details. Second, this representation of the circuit is easier for the computer to understand and thus, the system behavior can be simulated. This allows the designer to characterize many different implementations, so that a well-informed decision can be made. Third, a major source of errors in the design process can be attributed to conversion errors between the levels of the hierarchy. These errors can be reduced by using a computer to perform the necessary translations from the hardware description language. Some of the tools used in this phase are silicon com-

plers [Bray84a, Feld83a, Gajs84a, Sout85a, Youn85a] and floorplanning tools [Otte82a, Wime88a].

1.2. Logic synthesis and design

The logic synthesis and design phase provides a more refined view of the circuit than architectural design. This phase decomposes the high-level descriptions of the functional blocks obtained from architectural design into gates, such as, NANDs, NORs, FLIP-FLOPs, etc. This decomposition process can be performed independently for each of the functional blocks. Some of the key problems in this phase are state assignment for finite state machines, logic minimization, and logic simulation.

State assignment and logic minimization are an integral part of this phase, because they will decrease the area and increase the speed of the circuit. Logic minimizers allow the designer to use equations in forms that best describe the problem without having to worry about how the equation should be written to get the best speed and/or area. Many software tools have been developed to solve the state assignment problem [DeMi85a, Huer88a], and to perform logic minimization, namely, MINI [Hong74a] and ESPRESSO [Bray84b] for two level logic minimization, and MIS [Bray87a] and SOCRATES [DeGe86a] for multi-level logic minimization.

After the functional blocks have been decomposed into their constituent logic gates and blocks, the circuit is simulated with a logic simulator [Duml83a, Hajj83a] to verify the correctness of this design stage.

1.3. Gate/circuit design

In this phase of the design, the circuit is mapped into a particular technology. The basic components from logic design are converted into transistor implementations. The designer is concerned with the electrical aspects of the circuit, namely, power consumption, noise margins, and drive capabilities. Various transistor parameters, such as transistor lengths and widths are determined with the aid of circuit simulation programs, such as SPICE [Nage75a] or SLATE [Yang80a]. Recently, circuit optimization programs have been developed to assist the designer in determining the proper transistor sizes. Two approaches have been explored by researchers, namely, an expert systems based approach, iJADE [Lai87a], for optimizing general CMOS transistor networks, and iCOACH [Chen88a], a delay modeling approach, for optimizing dynamic CMOS circuits.

The mapping from logic blocks to transistors is guided by the layout style that will be used in the layout design phase. For example, if the layout is to be generated using a full-custom approach, very few restrictions are placed on the gate design. On the other hand, if a standard cell approach is used, or if the design is to be implemented in a gate-array, then the logic blocks are built only from cells that are in the cell library.

1.4. Layout design

The layout design phase takes the transistor networks developed in the circuit design phase and produces the geometrical shapes, which represent the various mask layers. There are several design approaches in this area, namely, full-custom, semi-custom, and gate array.

In the full-custom approach, very few restrictions are placed on the designer. Sub-circuit blocks and transistors configurations can be of arbitrary shapes and sizes. Various CAD tools have been developed to assist the engineer, such as, graphical editors [Oust84a, Tayl84a], sticks editors [Will78a], and geometric compactors [Doen87a]. Graphical editors are used to draw each of the rectangles, while sticks editors draw the circuit using line segments to represent the rectangles. The sticks editor can automatically generate design rule correct layouts from this simpler representation of the geometry. Geometric compactors have been developed to minimize the area of the layout by shifting, stretching, and squeezing the geometric features into a more optimized configuration.

The semi-custom approach, which has been very popular in application specific integrated circuits (ASICs), is more restrictive than the full-custom approach, but significantly reduces the design time. This reduction in design time is based on two premises, namely, using cells from previous designs saves time, and blocks of equally sized cells simplify cell placement. The first objective is achieved by restricting the designer to a set of predefined cells. These cells have been completely characterized and provide several different models for simulation. Each cell may have a logic model, a transistor model, a macro model, and a fast or slow transistor model for process variations. The second objective is realized by designing all the cells to be of equal height (the width varies with the complexity of the cell). With this restriction the cells can be efficiently placed in rows. The cells can then be efficiently interconnected using a channel router [Reed85a].

The main asset of the semi-custom approach, its cell library, is also its greatest liability. The cell library provides a significant reduction in the design cycle, but is only useful for the lifetime of the technology in use. In fact, the average cell library generally lags behind the processing technology by over one year [Koll88a]. The cost of regenerating a cell library for a new technology is expensive and dependent on the library size. Thus, from a maintenance viewpoint, the cell library should be as small as possible. However, the quality of the layouts increases with increasing library size [Keut87a]. Hence, most cell libraries are composed of simple gates, inverters, small AND-OR-INVERT (AOI) gates, small OR-AND-INVERT (OAI) gates, and several latches.

In addition to the above custom approaches, the designer may implement his design using gate-arrays [Naka80a, Ornd81a] or sea-of-gates [Noij85a, Saka85a, Ushi88a]. This technology has mainly catered to low volume designs. The gate-array and the sea-of-gates (also known as channel-less gate arrays) technology provides an even quicker turn-around time than the semi-custom approach. The gate-array wafer is a regular structure consisting of rows of equally sized transistors. These wafers are mass produced (up to metalization) and provide the foundation for all gate-array circuits. Thus, the advantage of the gate-array design is fast turn-around times because only the metalization layers need to be designed, since all other processing steps have already been performed. However, a disadvantage of the gate-array design is its poor performance compared to full-custom and semi-custom designs due to the gate-array's restriction to equally sized transistors.

1.5. Simulation and verification

In an ideal design environment, a final simulation and verification pass is not necessary. However, in today's design environment, the final simulation and verification pass plays a significant role in circuit design; it ensures that the circuit will behave as designed, and serves to uncover any oversights that may have developed during the design process. For instance, in a hierarchal design environment, the cells are developed independently. The complex interactions between the cells are hard to predict without the aid of simulation tools. Also, the various software tools used throughout the design are very complex and cannot be guaranteed to be error free.

The final simulation and verification pass used in circuit design is usually not very detailed, since the various subcircuits of the design have already been extensively simulated at many levels during each of the different design phases. This is desirable, since the sheer size of the circuit limits the simulation to logic simulation [Mess89a] or fast-timing simulation [Over89a].

1.6. Overview

The current approach to layout design has several deficiencies. Although the full-custom approach produces high quality layouts, it suffers from long design cycles. The semi-custom approach is too dependent on its cell library. The semi-custom approach cannot easily take advantage of advanced transistor minimization techniques which generate complex transistor configurations. For example, to implement the function

$$F = AD + ACE + BCD + BE,$$

requires only 10 transistors [Gee89a], as shown in Figure 1.2. However, a 2-level NAND

gate implementation requires 28 transistors. Although the semi-custom approach does produce denser layouts with larger cell libraries, the cell library cannot contain cells for all logic functions. In fact, maintaining a library of just AOI gates with at most 4 gates in series is prohibitive. Another problem in the layout design of logic devices has been the impact of parasitics on the circuit performance. Many speed-limiting delay paths are formed with long polysilicon features in the vertically stacked active circuit area with high fanout counts. These parasitics can be minimized with a second level of metal, by replacing the long high resistance polysilicon features with low resistance metal features, thus improving the circuit response.

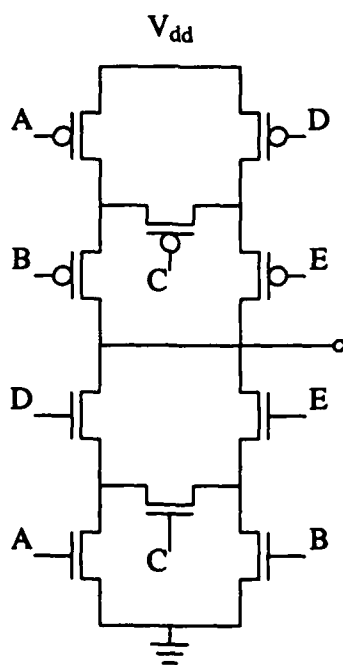


Figure 1.2. Complex Cell Design

This thesis presents an automatic circuit synthesis system which can take advantage of transistor minimization techniques for MOS circuits, such as applying switching network theory to minimize the number of transistors [Gee89a, Wu85a, Wu87a]. This system avoids the high cost of maintaining a large cell library [Keut87a] by generating the cells dynamically as needed. Since the cells are generated on demand, the synthesis system can use the latest design rules available, and the transistors within each cell can be sized individually to achieve the desired performance. The transistor circuit is partitioned into cells ranging up to 200 transistors in size, and laid out in the metal-metal matrix (M^3) layout methodology [Kang87a] using an automatic cell generator. The cells in the synthesis system are represented as incidence matrices. The incidence matrix is symbolically manipulated to minimize the number of rows and columns. The automatic cell generator will remove vertical constraints imposed on the nets by adding additional columns, so that cells can be generated to a specified maximum height. These constraints arise naturally from transistor drain-gate-source relationships (the drain-net of a MOS transistor must be assigned before the gate-net, and the gate-net assigned before the source-net) and from transistors connected in series. The matrix is then converted into a symbolic layout from which various physical layouts can be generated by specifying different design rules. These cells are then placed and interconnected with a standard cell placement and routing package.

This thesis is organized as follows. Chapter 2 discusses the features that automatic cell generators should possess in order to synthesize high-performance circuits. This chapter then compares various automatic cell generators, such as, PLAs, Wienberger

arrays, and gate matrices. This is followed with a description of the metal-metal matrix layout methodology, and a discussion on its advantages over other layout methodologies.

Chapter 3 presents symbolic matrix operations which are applied to the incidence matrices to minimize the area of the cells and to generate a symbolic layout. First, the impact of merging the columns in the incidence matrix on the physical layout is studied. Then, various symbolic matrix compaction methodologies are presented. This is followed by a detailed presentation of the approach used in this thesis. The presentation begins with a discussion on minimizing the total 1-1 distance, to be defined, to minimize the number of columns. Next, the columns are reordered to minimize the number of tracks necessary to lay out the circuit. Once the columns have been reordered, the nets are assigned to tracks and a symbolic layout is generated.

In Chapter 4, the issues pertinent to generating a physical layout from a symbolic representation are discussed. The symbolic layout system uses a variable grid-base approach and lays out circuits using the M^3 layout methodology. Unlike other symbolic layout systems, this system provides an additional level of abstraction. The layout symbols represent simple geometric primitives. The actual mask layers used to construct these primitives are determined automatically. This chapter begins with an overview of the symbolic layout system. Next, the layout primitives used in this system will be discussed as well as the techniques employed to minimize the area.

The next chapter, provides an overview of the entire synthesis system and provides comparisons with other layout techniques. Finally, Chapter 6 summarizes the results of this thesis and presents areas for future research in automatic circuit synthesis. The

Appendix provides the algorithm used in iDSIM [Over89a] for pass transistor detection, the input file for the 32-bit carry-look-ahead adder, a proof that minimizing the total 1-1 distance is NP-complete, and a user's manual for the synthesis system developed in this thesis.

CHAPTER 2.

BACKGROUND

In this chapter, various automatic cell generators and their deficiencies are explored. In general, the goal of automatic cell generators is to produce high quality layouts for circuits of various sizes and constraints. Some of the attributes in common to most cell generators are as follows: cells can be adjusted according to speed and power constraints, advances in technology can be implemented with little delay, and no time is wasted searching for cells in a large library. Currently, cell generators can be classified into two distinct architectures, namely, fixed and flexible.

In the fixed architecture cell generator, the layout pattern is fixed, and the logic functions are implemented by placing transistors or gates in predefined locations. This simplifies the layout process, but may hamper its ability to interface with other cells. The fixed architecture generally has no provision for controlling the input/output locations or for controlling the size and/or shape of the cells.

On the other hand, flexible architectures allow the user to control the input/output locations and the size and shape of cells, at the expense of using more complex layout algorithms. However, since all cells will be used in conjunction with other cells, many local objectives are not as important in the global optimization process. Hence, flexible architectures may be more advantageous on a global scale.

This chapter then closes with a presentation on an alternative layout style, namely, metal-metal matrix (M^3). The M^3 layout methodology is flexible and overcomes many of the deficiencies inherent in other styles. This style will form the basis for the cell generator in the iSITE system.

2.1. Programmable logic arrays (PLAs)

Programmable logic arrays provide a convenient and simple method for implementing two-level logic. They are regular structures consisting of two planes: the AND-plane and the OR-plane. The AND-plane forms all the minterms for the function and the OR-plane forms the concatenation of the minterms. Depending on the technology, the AND-OR planes are implemented as NAND-NAND planes or as NOR-NOR planes.

A typical PLA is shown in Figure 2.1. Notice that the OR-plane is identical in structure to the AND-plane except that it has been rotated 90 degrees. The PLA has a fixed architecture. Each row of the AND-plane represents one minterm. The input and output signals form the columns of the PLA, which run through the entire cell. Transistors are located only at the grid points formed at the intersection of the input/output signals and the minterms.

Most PLAs are very sparse, so that the straightforward implementation described in the previous paragraph would waste a significant amount of area. The area of a PLA is proportional to the number of columns multiplied by the number of rows, which is equal to the product of the number of input/output signals and the number of minterms. So, to reduce the area of a PLA, the number of rows and columns should be minimized. There are two methodologies for achieving this objective, namely, logic minimization to

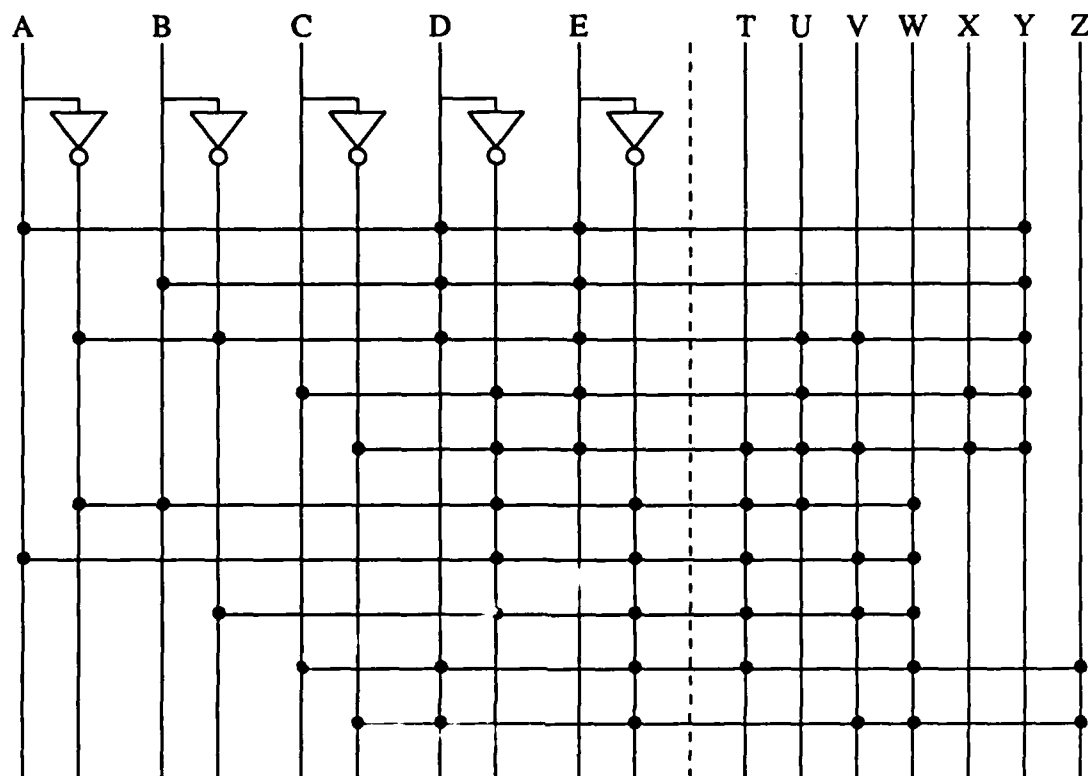


Figure 2.1. Programmable Logic Array

minimize the number of minterms, and hence rows, and folding, which can be applied to both the rows and the columns.

Almost all logic minimization tools in use today are based on heuristic methods, since an optimal method would require $O(\frac{1}{n}C^n)$ time [Bray84c]. Two heuristic programs used for logic minimization are MINI [Hong74a] and ESPRESSO [Bray84b]. The MINI program finds a near minimal solution by iteratively improving an initial solution. The cost function is simplified by assigning each implicant a constant cost instead of a cost proportional to its size. Each iteration consists of three steps: enlargement,

reduction, and reshaping. The ESPRESSO algorithm is very similar to the MINI algorithm. It also uses an iterative improvement algorithm. The cost function used by ESPRESSO is based on the number of product terms and the number of literals. The main loop in the ESPRESSO algorithm consists of expanding each of the terms, finding an irredundant cover, and reducing the cover. Both of these programs try to minimize the number of implicants. Neither program considers minimizing the complemented form of the function, or considers the foldability of the resulting cover.

Folding columns (rows) is simply allowing two columns (rows) to share a single column (row). Of course, not all columns (rows) of a PLA can be folded. Before two columns (rows) can be folded they must be disjoint. Also, folding introduces other constraints. For instance, folding column i on top of column j forces all the rows containing signal i to be above all the rows containing signal j , because two signals in a PLA cannot be intermixed. Finding the optimal folding set is an NP-complete problem. However, optimal algorithms [Hwan86a, Lewa84a] are used for simple-folding because parasitic losses and delays limit the size of PLAs. Heuristic algorithms based on bipartite folding [Egan82a], or on graph theory [Hach82a], or on simulated annealing [Deva86a] have also been developed for simply-folded and multiply-folded PLAs.

The PLA can be generalized to implement sequential logic by simply adding latches, as shown conceptually in Figure 2.2. This structure, known as the storage logic array [Smit82a], embeds both the AND and the OR operations in one plane and allows higher level constructs such as latches to be implemented at the grid points.

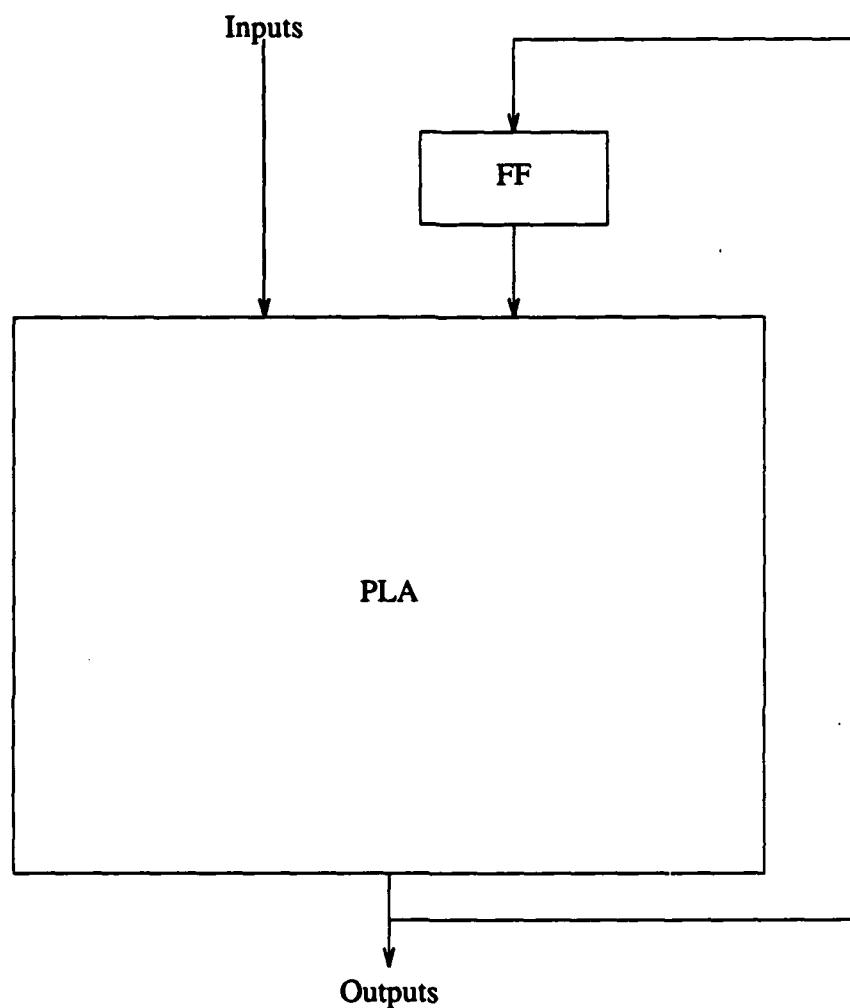


Figure 2.2. Conceptual Diagram of Storage Logic Array

Programmable logic arrays have many deficiencies. Even with folding, PLAs tend to grow quadratically with respect to input size. They are also very inefficient for some types of circuits. A PLA may implement an n -input function with as many as $O(2^n)$ product terms. Multi-level logic can be implemented with PLAs, if the logic is partitioned into a series of two-level logic blocks. However, PLAs cannot be used to implement

bridge circuits or other complex transistor configurations generated from transistor minimization techniques. Also, folding the rows of a PLA places restrictions on the columns of the PLA, and hence, on the input/output signals. Thus, the pin positions of these signals may not be controllable by the designer. Finally, the designer has little control over the aspect ratio of the cell generated.

2.2. Weinberger array

The Weinberger array [Wein67a] is not restricted to two-level logic as in the PLA. It can implement multi-level Boolean functions using NAND or NOR logic. In the Weinberger array, the input/output signals run horizontally in metal, while the diffusion runs vertically. Transistors are created by overlapping the polysilicon layer over the diffusion and connecting the polysilicon to the metal lines. Each column in the Weinberger array represents a single gate. An example of a Weinberger array is shown in Figure 2.3.

All the gates in the Weinberger array are of the same height, independent of the number of transistors in each gate. However, columns and rows can be folded to minimize the area. A drawback to this style is that it was designed for NMOS technology and is not easily adaptable to CMOS circuits. Another shortcoming of the Weinberger array is that it suffers from high parasitic RC delays due to long diffusion runners. This methodology, like the PLA, cannot implement arbitrary transistor configurations.

2.3. Gate matrix

The gate matrix layout style [Lope80a] is even less restrictive than the Weinberger array. This style can lay out any transistor interconnection pattern. The gate matrix

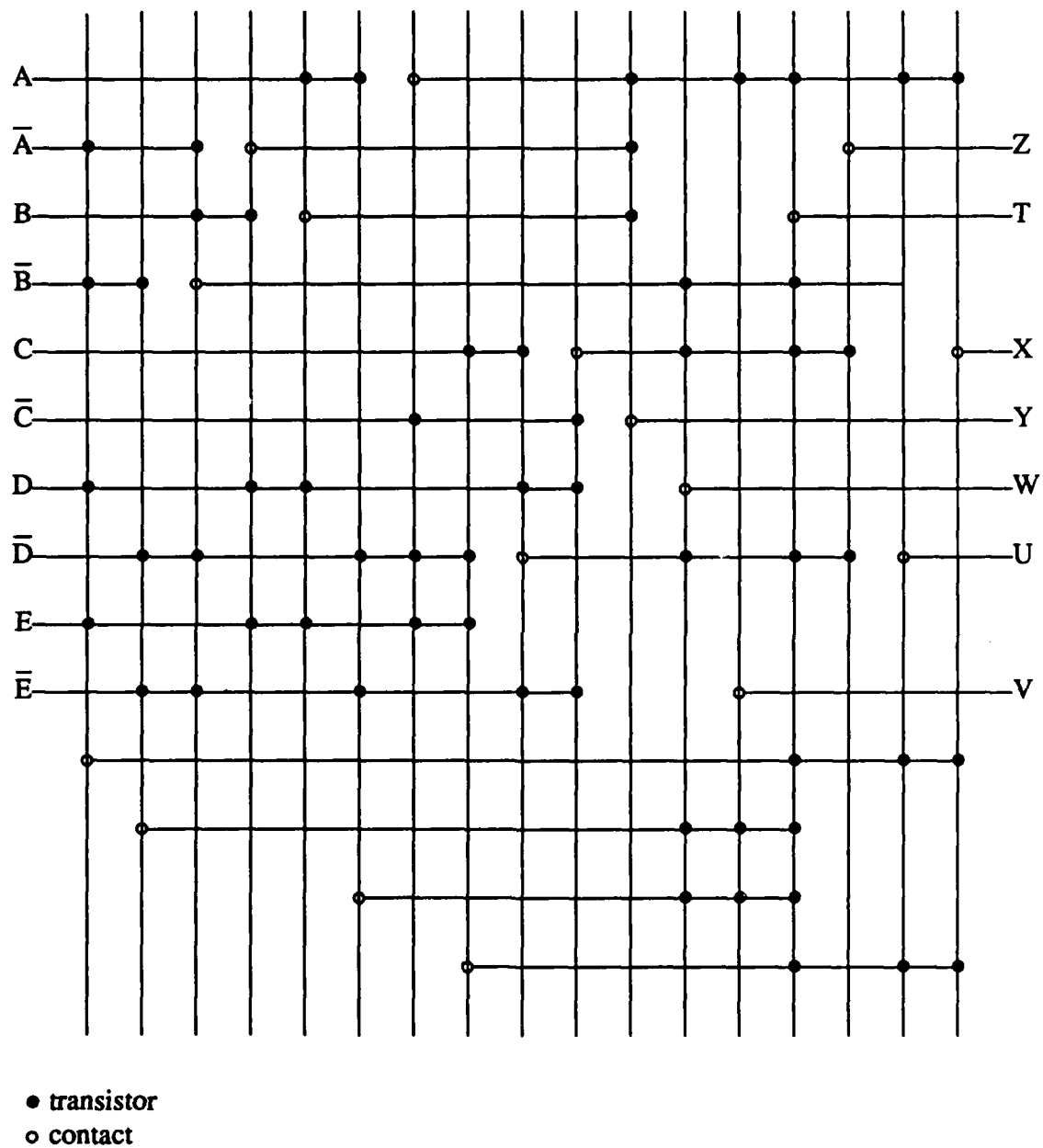


Figure 2.3. Weinberger Array

layout style separates all the transistors of the same type into two regions. The input/output signals are equally spaced and parallel and run vertically through the cell in polysilicon. All the transistors driven by the same gate signal are located in the same column underneath the transistors' input signal. In each row of the matrix, the sources and drains of transistors are interconnected in a series or parallel fashion using horizontal metal lines. Vertical diffusion runs may be used to interconnect the sources and drains of transistors in different rows. A typical gate matrix layout is shown in Figure 2.4.

To minimize the area of gate matrix layouts, an ordering of the columns must be found to minimize the number of rows necessary to lay out the circuit. This problem has been shown to be NP-complete, so a number of heuristic methods have been developed, namely, interval graphs [Ohts79a, Wing85a], min-cut with dynamic nets [Hwan87a], Euler paths [Chen88b], and simulated annealing [Leon86a].

The interval graph heuristic is based on the premise that the number of tracks necessary to lay out a set of unconstrained intervals without overlapping the intervals on the same track is equal to the size of the largest clique in its corresponding interval graph [Berg70a]. Since each connection graph is a subgraph of some interval graph, the problem is to find an interval graph with a minimum clique size which contains the connection graph. A heuristic algorithm has been developed by Ohtsuki, et al. [Ohts79a], which constructs several different augmentations and chooses the interval graph with the smallest clique. This method has been applied to gate matrix layouts by Wing, et al. [Wing85a].

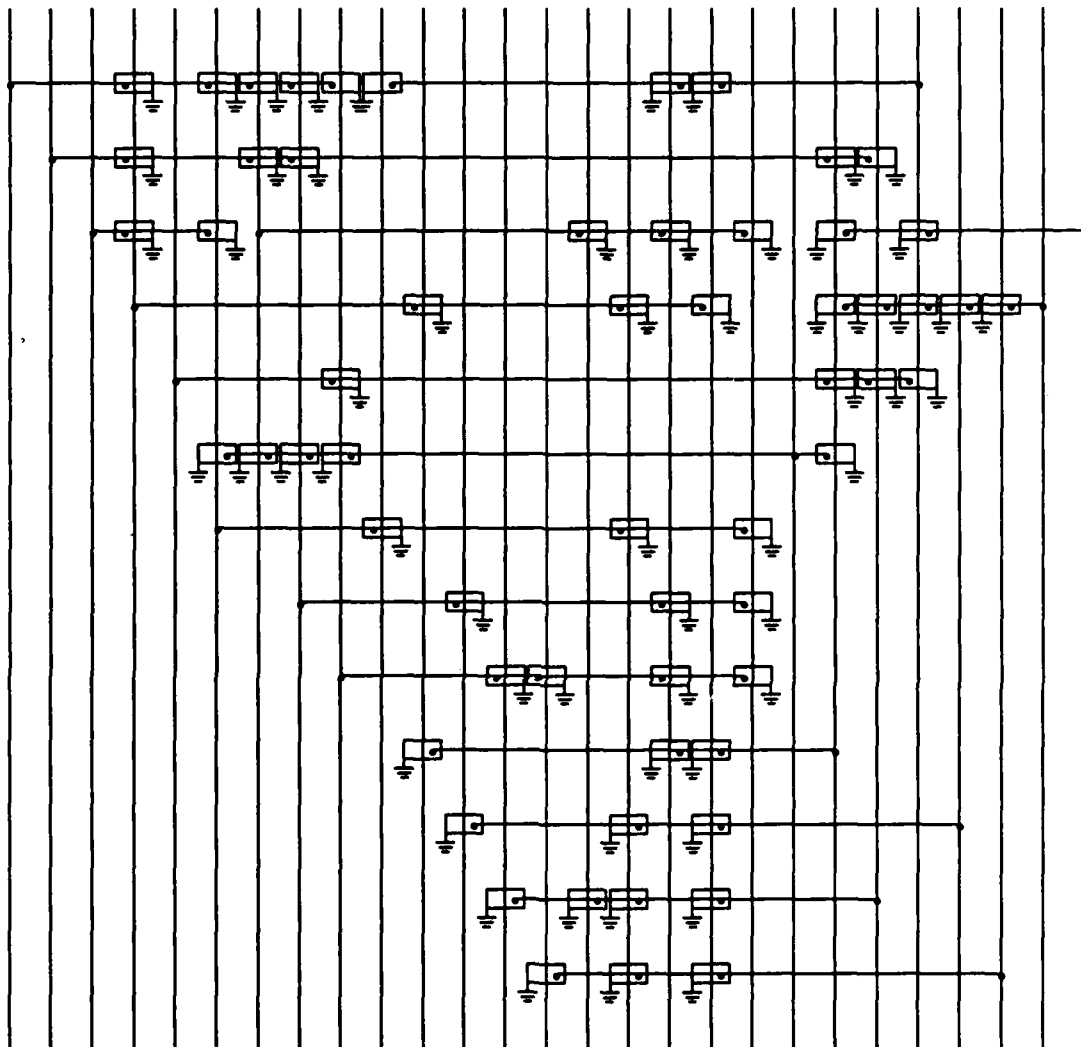


Figure 2.4. Gate Matrix

The min-cut method with dynamic net-lists [Hwan87a] minimizes the number of tracks by minimizing the net density, since the net density is a lower bound on the number of tracks necessary to lay out the circuit. The dynamic net-list concept is used to

allow the net binding to be delayed until the signal ordering has been determined, thus avoiding many sub-optimal configurations.

The Euler path method [Chen88b] has been used to lay out complex gates in the gate matrix layout style. The circuit is represented as a multigraph, where the edges represent the drain/source diffusion area of transistors, and vertices represent the drain/source connection of the transistor. A path which traverses every edge once in this graph is an Euler path. The column order derived from this path is optimal. This method can be generalized to circuits with several gates. However, an optimal ordering will be found only if an Euler path exists. Heuristics can be developed to find a minimal set of edges to augment the multigraph so that an Euler path exists.

Simulated annealing has also been applied to the gate matrix area minimization problem. Many other heuristic algorithms formulate this minimization problem as a series of independent sub-problems. Leong [Leon86a] presents a cost function that minimizes the area and the total wire length, with moves that simultaneously consider gate permutation, dynamic binding, and net merging.

The gate matrix layout style is deficient in two areas, namely, long polysilicon features in large cells, and power and ground distribution. The long polysilicon features can result in large parasitic RC delays. These delays can be reduced by using an additional layer of metal in parallel with the polysilicon lines. However, extra contacts which are difficult if not impossible to add, must be added to electrically connect these layers in the congested areas of the gate matrix without increasing the area further. Also, this configuration results in additional parasitic capacitance in the signal lines. The other

problem with the gate matrix layout, the power and ground distribution, is currently solved with long diffusion runners. However, this is unsatisfactory because of the high parasitic resistance and capacitance associated with diffusion runners. If the power and ground is routed entirely in metal, the area can be enlarged by as much as 45 percent.

2.4. Metal-metal matrix (M^3) layout

The trend of the future generation of CMOS technology is toward faster circuits using smaller features and reduced power supply voltage. Reducing the power supply voltage may require better control over the threshold voltage for both the P-type and the N-type transistors. Controlling the threshold voltage by using ion-implantation and a single type of polysilicon (Figure 2.5(a)) is vulnerable to interdiffusion problems at finer design rules. A more desirable method would be to form transistors using both P-type and N-type polysilicon (Figure 2.5(b)) and interconnecting the gates with metal at a later processing step.

The two layers of aluminum available in CMOS technology for logic devices has mainly been used for intercell routing. However, even at a moderately high operating frequency, the speed-limiting delay paths are often formed within the cell in the vertically stacked active circuit area with high fanout counts. Two-level metal technology can be used to solve these timing bottlenecks by replacing long polysilicon features with aluminum features. The metal-metal matrix (M^3) methodology [Kang87a] provides a globally structured layout method for one-layer polysilicon and two-layer aluminum MOS technology. It employs maximal use of metal interconnections to minimize the length of diffusion and polysilicon runners.

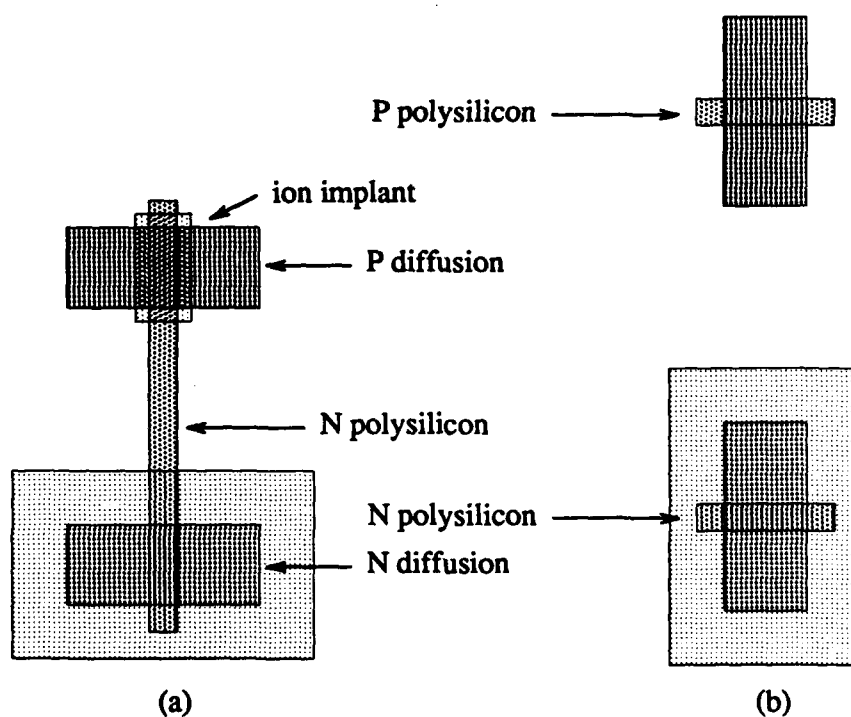


Figure 2.5. Controlling the Threshold Voltage in Transistors
(a) A Single Polysilicon Runner May Have Interdiffusion Problems
(b) Separate Polysilicon Features Allows the Threshold Voltage to be Controlled Independently

The main objectives of the M^3 layout methodology are as follows:

1. minimize the length of the polysilicon runners to suppress large parasitic RC delays,
2. allow the threshold voltages of PMOS and NMOS transistors to be controlled independently without interdiffusion problems by disallowing vertical stacking of PMOS and NMOS transistors with a contiguous strip of polysilicon,

3. allow one contiguous strip of polysilicon between transistors of the same type to accommodate multiple fanouts,
4. make all other interconnections with aluminum runners to minimize parasitic RC delays,
5. make all the terminals of modules strictly in metal to facilitate all metal inter-cell routing to avoid long polysilicon features in the routing channel.

The above objectives can be realized by forming a matrix with both layers of aluminum. In the M^3 style, as shown in Figures 2.6(b) and (c), all signals run vertically in metal to form the *signal columns*. Interspersed between these metal lines are *diffusion columns*. The other layer of metal is used to horizontally interconnect the sources and drains of transistors, to run power and ground busses, and to reduce the length of long polysilicon runners between the transistor gates and their corresponding signal columns. Short polysilicon runners between transistor gates and signal columns are permitted. These horizontal nets can be folded to reduce the layout area. The power and ground busses are usually assigned to the first and last track of the cell to facilitate chip level power and ground distribution.

In this implementation of the M^3 layout style, two variations can be generated automatically. A circuit schematic and its layouts are shown in Figures 2.6(a), (b), and (c). The first style is aimed at increasing the packing density of circuits that are not on a critical path, where area constraints are the primary factor. The second style is aimed at implementing cells in a critical path, where delay constraints are the main concern.

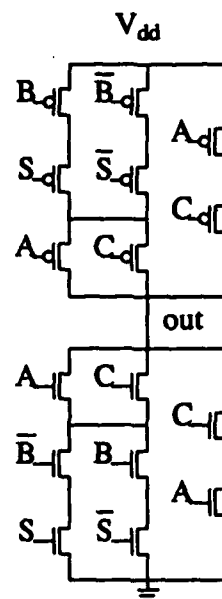


Figure 2.6. Metal-Metal Matrix
(Continued on next page)
(a) Circuit Schematic

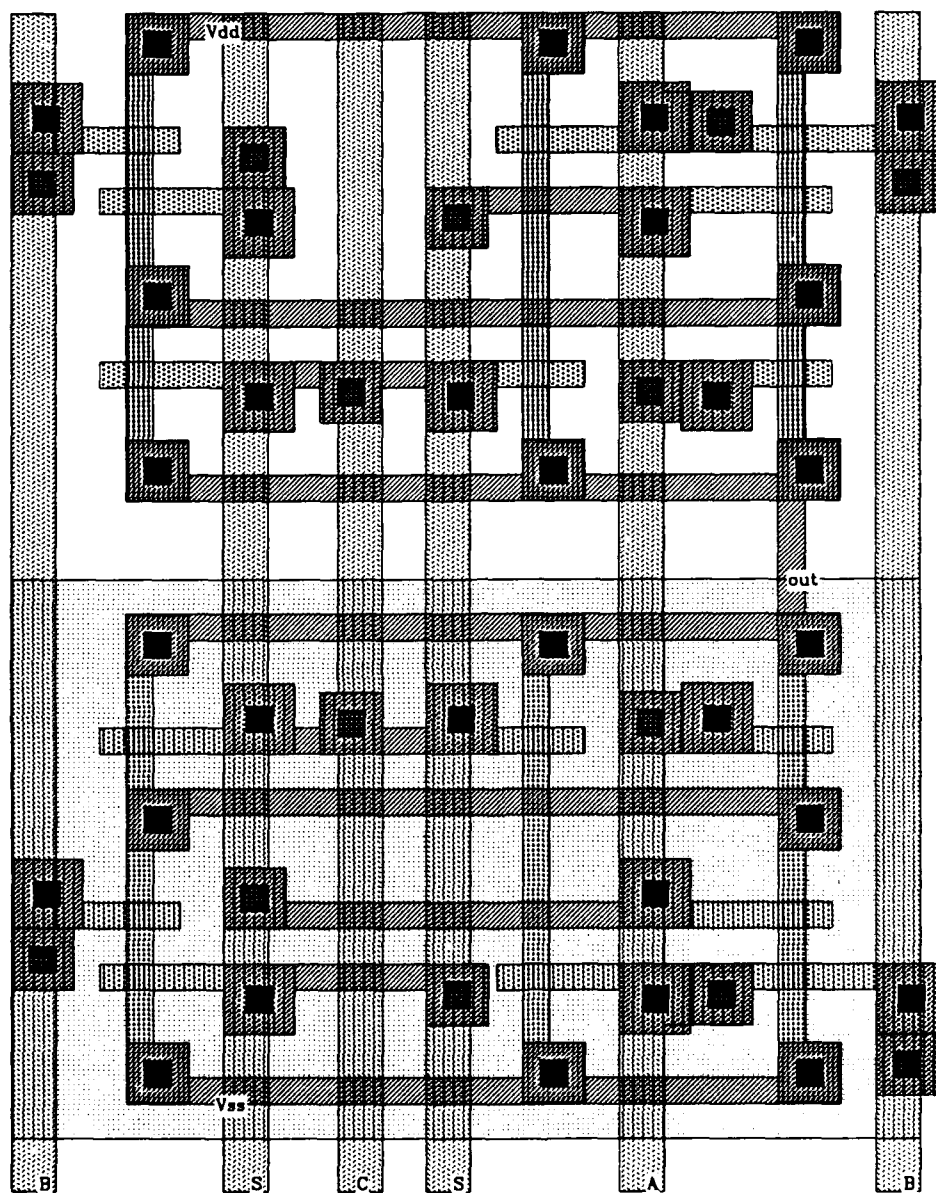


Figure 2.6 (Continued)
(b) M^3 Style 1

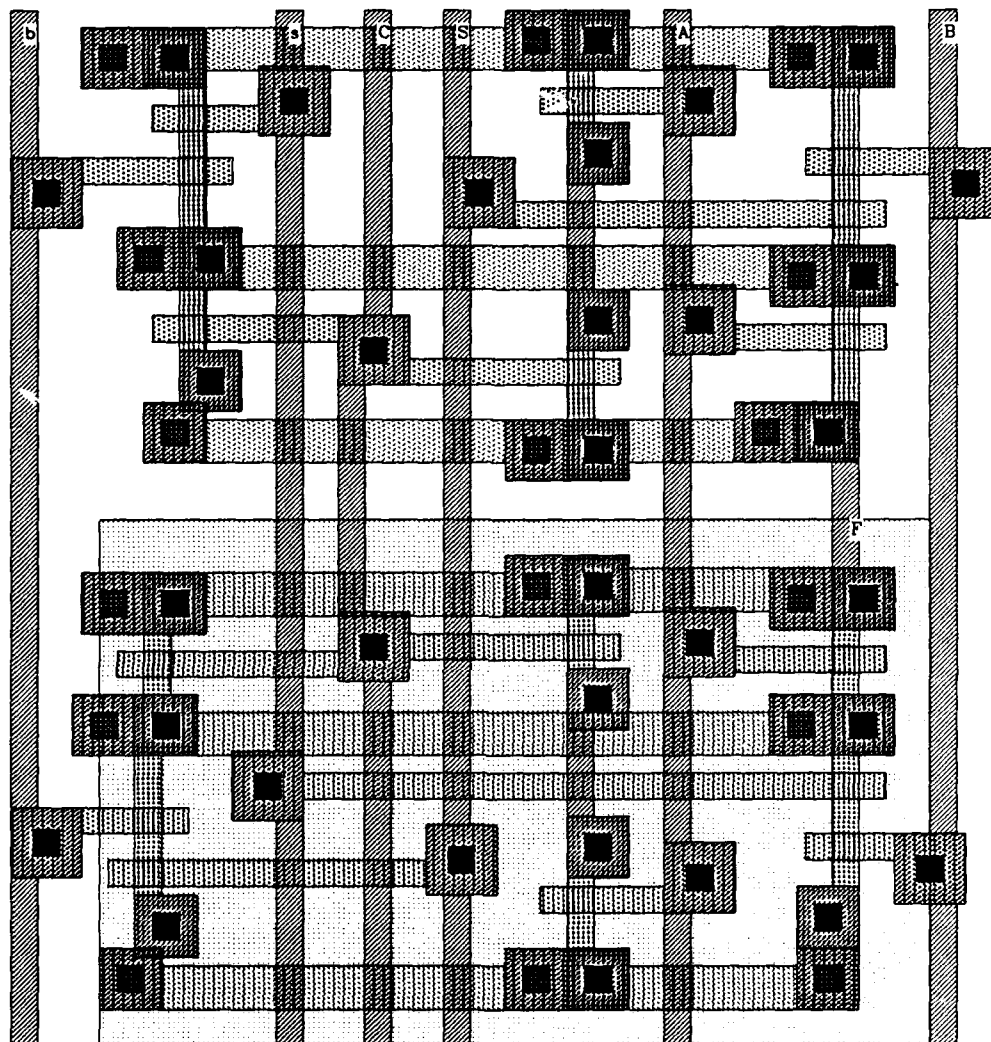


Figure 2.6 (Continued)
(c) M^3 Style 2

In the first style, shown in Figure 2.6(b), the first level (lower) metal is used to route the horizontal metal interconnections and the second level (upper) metal is used for the vertical metal interconnections. This implementation of the M^3 layout style generally produces circuits smaller in area than the second implementation (to be discussed next), since the area underneath the signal columns can be used to accommodate large transistors, as well as the metal-poly contacts needed to electrically connect the gates of transistors. However, large cells in this style can suffer from high parasitic resistances and capacitances due to long diffusion runners.

In the second style shown in Figure 2.6(c), the functions of the two metal lines are exchanged. In other words, the first level metal lines are used to form the vertical metal interconnections, and the second level metal lines are used for the horizontal metal interconnections. This scheme allows the long diffusion runners to be easily replaced with metal, and the parasitic resistances and capacitances are significantly reduced making this style even more suitable for high-speed circuits.

Although two different variations of the M^3 layout style have been presented, the same area minimization strategy without any modifications can be applied to both variations. Both styles have similar structures, and hence, can be represented by one symbolic layout, if layer-independent symbols are employed, as in the iSILVER system [Gee88a].

CHAPTER 3.

SYMBOLIC CIRCUIT REPRESENTATION AND GENERATION

3.1. Incidence matrices

A MOS circuit can be represented as a graph or by its incidence matrix. In the graph representation, Figure 3.1 (b), each edge corresponds to a transistor and similarly labeled edges in the graph represent transistors driven by the same signal. The incidence matrix is constructed by representing each node in the circuit by a row. Then, for each transistor in the circuit, a column is inserted in the incidence matrix. This column is labeled with the gate signal and has two row entries of $1s$ corresponding to the source and drain nodes of the transistor. In this implementation, the rows corresponding to power and ground are restricted to the first and to the last rows of the matrix to facilitate inter-cell power and ground routing. The incidence matrix for the circuit in Figure 3.1 (a) is shown in Figure 3.2. For NMOS circuits the single load transistor for each channel-connected component can be easily implemented in the top portion of the layout and is omitted in the following discussion.

3.2. Incidence matrix compaction

The incidence matrix for a circuit is generally very sparse. This sparsity can be reduced by merging columns. Merging two columns together in the incidence matrix corresponds to placing the two transistors represented by the columns in the incidence

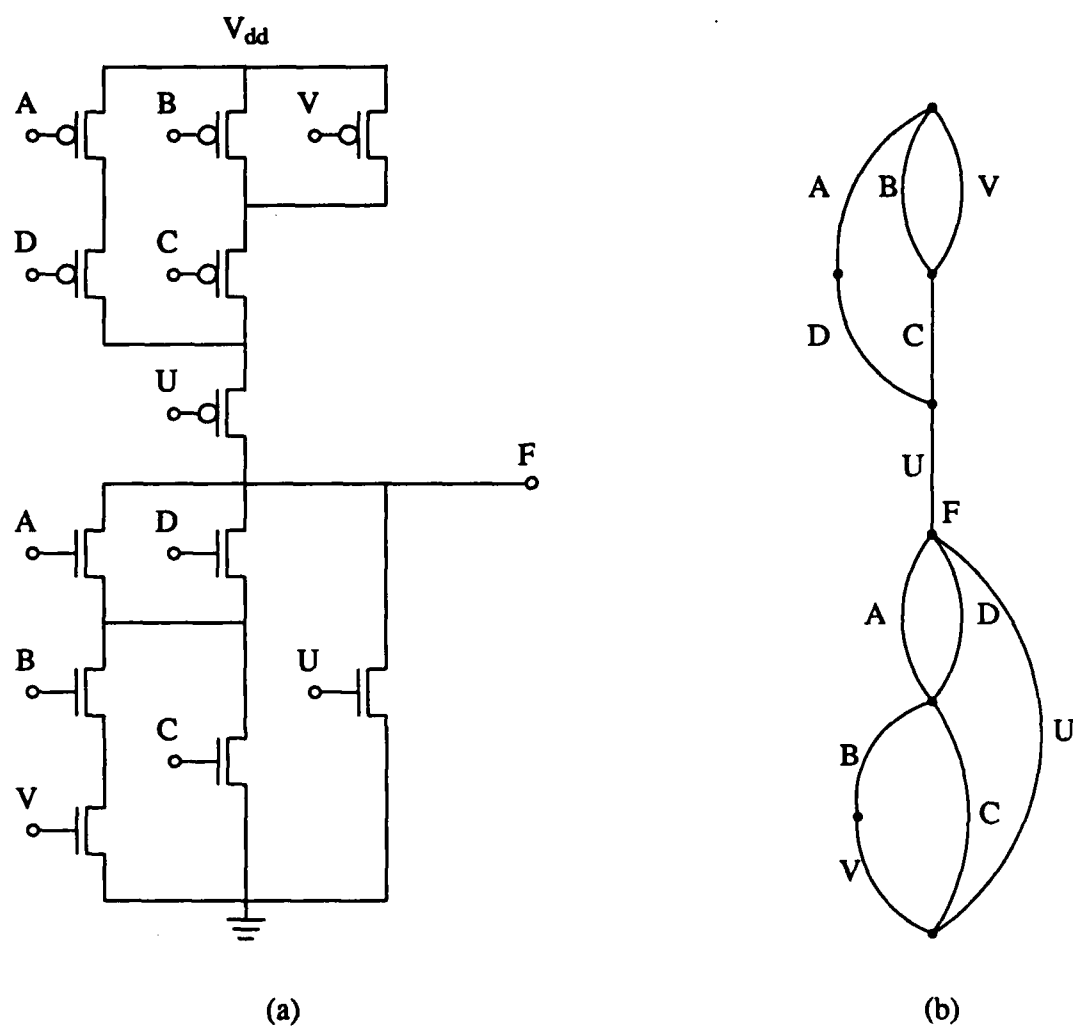


Figure 3.1. Circuit Schematic and its Circuit Graph

(a) Circuit Schematic

(b) Circuit Graph

	A	B	V	D	C	U	A	D	U	B	V	C
V_{dd}	1	1	1	0	0	0	0	0	0	0	0	0
	1	0	0	1	1	0	0	0	0	0	0	0
	0	1	1	0	1	1	0	0	0	0	0	0
F	0	0	0	1	0	1	1	1	1	0	0	0
	0	0	0	0	0	0	1	1	0	1	0	1
	0	0	0	0	0	0	0	0	0	1	1	0
V_{ss}	0	0	0	0	0	0	0	0	1	0	1	1

Figure 3.2. Incidence Matrix

matrix in the same physical column. There are several strategies available for merging the columns in CMOS circuits, namely, unrestricted NMOS and PMOS merging, unrestricted NMOS-PMOS transistor pair merging, and merge only transistors of the same type.

In the unrestricted NMOS and PMOS merging, no distinctions are made between the transistor types. Transistors may be merged together if they do not overlap. This methodology will produce the smallest symbolic layout, but is undesirable because this methodology tends to generate many small well regions.

The second approach, merging NMOS-PMOS transistor pairs, minimizes the number of well regions by pairing each NMOS transistor with a PMOS transistor,

forming alternate rows of NMOS and PMOS transistors, as shown in Figure 3.3(a). To decrease the number of wells further, every other pair can be mirrored as in Figure 3.3(b). Since most signals in CMOS circuits drive (or are driven by) an equal number of NMOS and PMOS transistors, the span of these signal lines can be significantly reduced. Hence, the number of transistors and signals that can be folded together is increased.

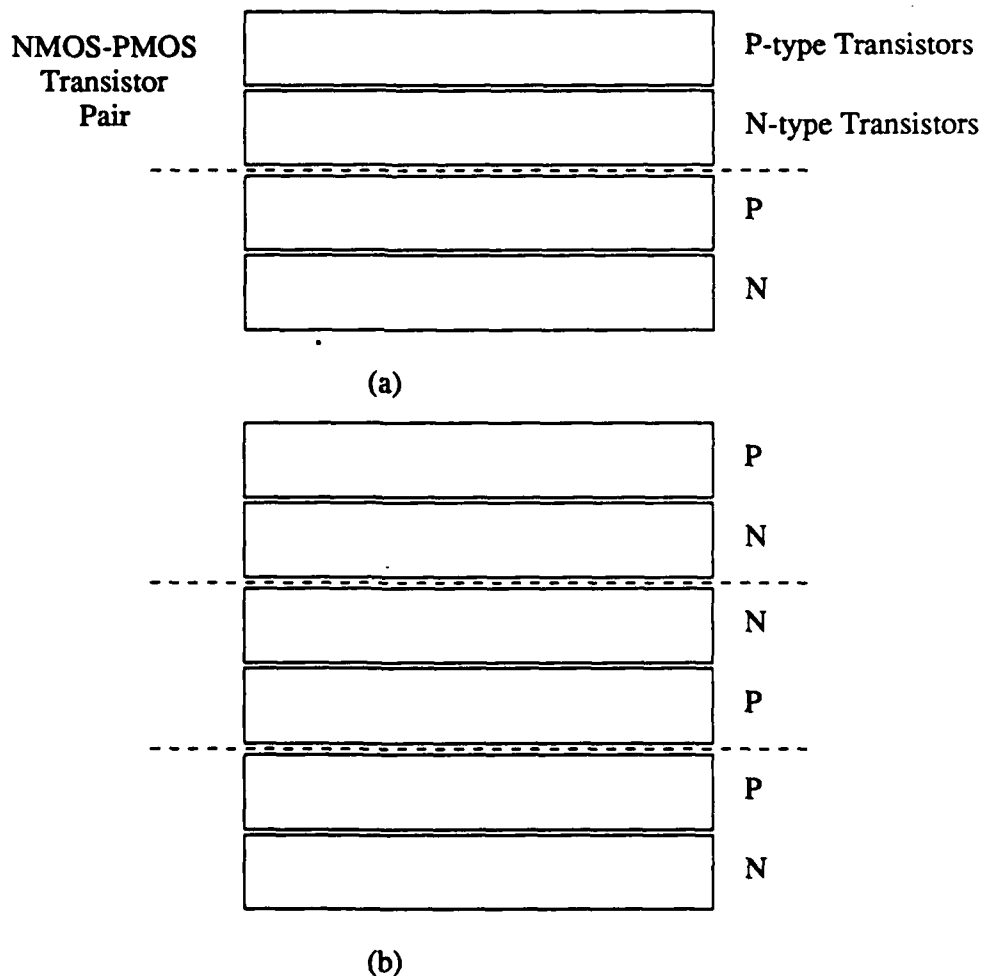


Figure 3.3. Column Folding Methodologies
 (a) Alternate PMOS-NMOS Transistor Row Layout
 (b) Mirroring PMOS-NMOS Row Pairs to Merge Well Regions

The final approach, merging only transistors of the same type, allows all the transistors of a given type to be placed in a single well. However, most signals cannot be merged together, because of overlapping spans. Although the second approach may achieve smaller symbolic layouts because the signal columns can be folded, it also incurs a higher area penalty from having additional wells. The average physical row height for the second approach is given by

$$H_2 = W_{\text{tran}} + S_{\text{tran,guard}} + W_{\text{guard}} + S_{\text{well,well}}$$

for a twin tub CMOS process (shown in Figure 3.4), where

$S_{\text{tran,guard}}$ = separation between transistor and guard ring,

$S_{\text{well,well}}$ = well to well separation,

W_{guard} = width of guard ring,

W_{tran} = transistor height (includes source and drain).

However, the average row height for the third approach is given by

$$H_3 = W_{\text{tran}} + \frac{2 * (S_{\text{tran,guard}} + W_{\text{guard}} + S_{\text{well,well}})}{N_{\text{rows}}}$$

Assuming,

$$S_{\text{tran,guard}} = L,$$

$$S_{\text{well,well}} = L,$$

$$W_{\text{guard}} = L,$$

$$W_{\text{tran}} = 3L,$$

where L is the minimum transistor length. The ratio of the row heights is

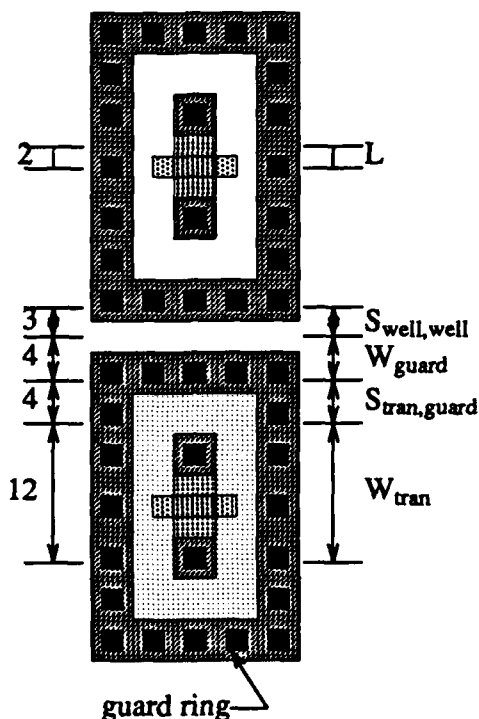


Figure 3.4. Feature Separation for 2 micron Process

$$\frac{H_2}{H_3} = \frac{6L}{3L + \frac{2 * (S_{\text{tran,guard}} + W_{\text{guard}} + S_{\text{well,well}})}{N_{\text{rows}}}} \approx 2.$$

Thus the second approach will be advantageous only when the area saved by folding signal columns results in over a factor of two area savings. In this thesis, only the columns representing transistors of the same type will be folded.

3.2.1. Symbolic matrix compaction methodologies

To achieve a compact layout, the sparsity of the A_n matrix is first reduced by merging columns, and the number of horizontal tracks is reduced by reordering the merged columns. The objective of this symbolic matrix compaction is to find a permutation of both the rows and the columns which permits a maximal set of logical columns (and rows) to be implemented in the same physical column (or row). However, to find the minimal M^3 layout by inspecting all possible permutations requires time exponential in the number of transistors and the number of signals. This problem can be solved either as a two-dimensional optimization problem or as a sequence of two simpler one-dimensional problems.

A number of methods for performing symbolic matrix compaction such as simulated annealing, folding, or interval-graph based techniques have been used in the past. A two-dimensional symbolic matrix compaction method has been developed by Devadas and Newton [Deva86a] using simulated annealing. This method for symbolic matrix compaction produces slightly denser layouts than folding, but requires much more CPU time.

Folding techniques have been studied extensively and applied mainly to PLAs [Hach82a, Hwan86a, Lewa84a]. Algorithms exist for optimal simple-folding using branch-and-bound algorithms [Lewa84a] as well as heuristic techniques for multiple-folding [Hwan86a], mixed row-and-column folding [Hach82a], and simple folding on compacted incidence matrices [Gee87a]. The method by Hachtel, et al. [Hach82a] for mixed folding is to fold rows and columns alternately. These algorithms are one-

dimensional compaction algorithms and their running times range from exponential to $O(n^3)$.

Another technique for symbolic matrix compaction employs interval graphs. This method is based on the fact that the number of tracks necessary to lay out a set of unconstrained intervals without overlapping the intervals on the same track is equal to the size of the largest clique in its corresponding interval graph [Berg70a]. Thus in this method the layout problem is represented by a connection graph. Since each connection graph is a subgraph of some interval graph, the problem is to find an interval graph with a minimum clique size which contains the connection graph. A heuristic algorithm has been developed by Ohtsuki, et al. [Ohts79a] which constructs several different augmentations and chooses the interval graph with the smallest clique. This method has also been applied to the gate matrix layout by Wing, et al. [Wing85a] and to PLAs by Yu and Wing [Yu85a].

Although the compaction problem, posed either as a two-dimensional optimization problem or as two one-dimensional problems, is NP-Complete, a two one-dimensional approach is chosen, mainly because each one-dimensional problem is much smaller in size than a two-dimensional one, which is an important factor for exponential time algorithms. This heuristic approach does not guarantee the globally minimum-area solution, but it leads to competitively dense M^3 layouts. The motivation for developing heuristic algorithms is twofold. First, heuristic algorithms can be used to obtain an upper bound in a short time. This upper-bound can significantly improve the computation time of the branch-and-bound type algorithms by allowing inferior solutions to be pruned earlier.

Second, the resulting layouts generated are often *good enough* for the problem at hand. This algorithm minimizes the net density which is a lower bound on the number of tracks required in the layout. The time complexity of this algorithm is shown to be $O(n^2)$.

3.2.2. Column merging

Definition: Total 1-1 distance

There are exactly two 1s in each column of an incidence matrix; the remaining row entries are all zeros. The distance between the two 1s in each column as measured by the difference in their row numbers is called the *1-1 distance*. The sum of 1-1 distances of all columns is defined as the *total 1-1 distance* of the incidence matrix. \square

The incidence matrix is not unique for each circuit. The rows of the matrix can be permuted to obtain another equivalent matrix. The total 1-1 distance is related to the number of symbolic area units required to lay out the circuit. The total 1-1 distance of the incidence matrix in Figure 3.2 is 11. The rows of the incidence matrix may be permuted to minimize the total 1-1 distance. However, finding the minimum total 1-1 distance of an incidence matrix is an NP-complete problem.

A heuristic procedure is used to minimize the total 1-1 distance as follows. The incidence matrix is first represented by a graph where each row corresponds to a node and each column to an edge. The two nonzero entries in each column correspond to the terminal nodes of the edge. Notice that if the n nodes are required to lie in a straight line, the total number of edges cut by the $n-1$ cut lines placed between each of the $n-1$ consecutive pairs of vertices is equal to the total 1-1 distance of a matrix with the

corresponding row order. Therefore, minimizing the total number of edges cut by the cut lines between vertices is equivalent to minimizing the total 1-1 distance of the matrix.

This algorithm minimizes the total number of edges cut by recursively applying a min-cut algorithm to the graph. The graph is partitioned into two subgraphs, such that the number of edges with terminals in both subgraphs is minimized. Before each of these subgraphs is partitioned, all the nodes to the right (left) of the current subset of nodes are represented as a single node to be called a pseudo node. The new partition with the right (left) pseudo node will be placed on the right (left), since this partition is in some sense more strongly connected to the right (left) side. Therefore partitions with both pseudo nodes in the same subgraph are not allowed. Each subgraph is recursively partitioned until the subgraphs are *small enough*. For this implementation, a subgraph is considered small enough when it consists of one real node.¹

The columns of the incidence matrix are now merged to form a column-compacted matrix by using the left-edge-first algorithm [Hash71a]. Figure 3.5 shows the process of partitioning the NMOS portion of the graph in Figure 3.1. Recall that in M^3 layout, power and ground busses are assigned to the first and last horizontal tracks. So without loss of generality, assign the V_{ss} node to be the left pseudo node (the V_{dd} node is assigned to the right pseudo node). The total 1-1 distance is now reduced from 11 to 9,

¹ This heuristic algorithm can be coupled with an optimal algorithm to minimize the total 1-1 distance. For instance, a dynamic programming approach which computes all the intermediate states can solve the minimum total 1-1 distance problem in $O(n^2 2^n)$ time. So, if this dynamic programming approach is coupled with this heuristic algorithm, then subsets with less than 20 nodes can be considered *small enough*. The order of this coupled algorithm is still $O(n^2)$ since the time necessary to compute the total 1-1 distance for a *fixed* sized subset is constant. However, for small n , this algorithm would still behave like the dynamic programming algorithm, $O(n^2 2^n)$, and exhibit the $O(n^2)$ behavior only for large n .

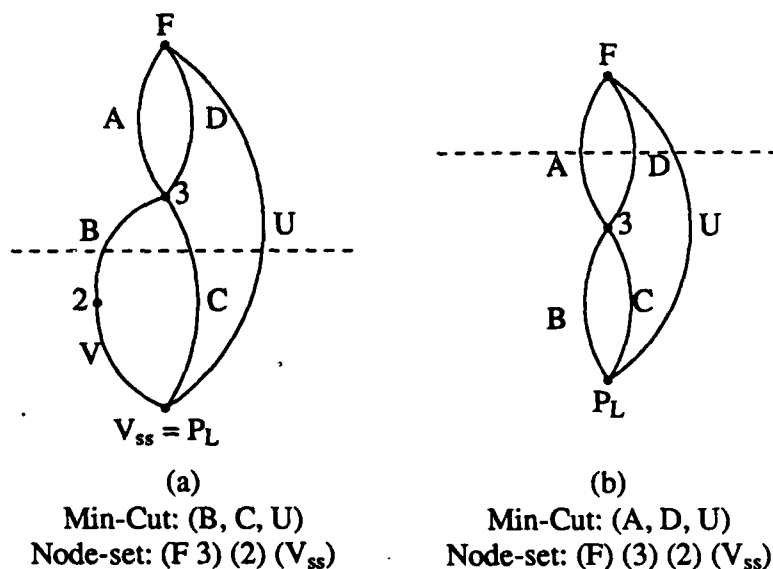


Figure 3.5. An Example of Min-Cut Ordering for Minimizing the Total 1-1 Distance

which in this case is the minimum value with the V_{ss} node restricted to the last row in the incidence matrix. The resulting compacted incidence matrix is shown in Figure 3.6. In the compacted incidence matrix notation, a "2" represents an overlap of two "1"s.

The min-cut procedure to partition the graph into two parts can be done in time $O(n^2)$. Therefore the time necessary to recursively partition the graph is

$$\sum_{i=0}^{\log_2 n} 2^i \left[\frac{n}{2^i} \right]^2 \leq 2n^2.$$

The matrix can be reordered in time $O(n)$ and the columns are then merged in time $O(n \log n)$. Thus, the total running time for the column compaction is $O(n^2 + n \log n + n) = O(n^2)$.

	A	D	U
	B	C	
	V		
F	1	1	1
	2	2	0
	2	0	0
V _{ss}	1	1	1

Figure 3.6. Column Compacted Incidence Matrix

3.2.3. Row merging

The number of tracks in the layout can be minimized by allowing several non-overlapping nets to share the same track. The net density of the cell is a lower-bound on the number of tracks required to lay out the circuit. However, the net density is dependent on the order of the signal and diffusion columns, as shown in Figure 3.7.

One possible algorithm for determining a column order which minimizes the number of tracks could be based on the min-net-cut algorithm of Kernighan and Lin [Kern70a]. To reduce the number of horizontal tracks, a netlist from the column-compacted matrix would be generated, and then the K-L min-net-cut algorithm would be applied to the netlist to partition the columns into two sets, such that the number of nets with terminals in both sets is minimized. Two pseudo nodes representing all the terminals to the right and to the left of each subset are appended to each half. Then this

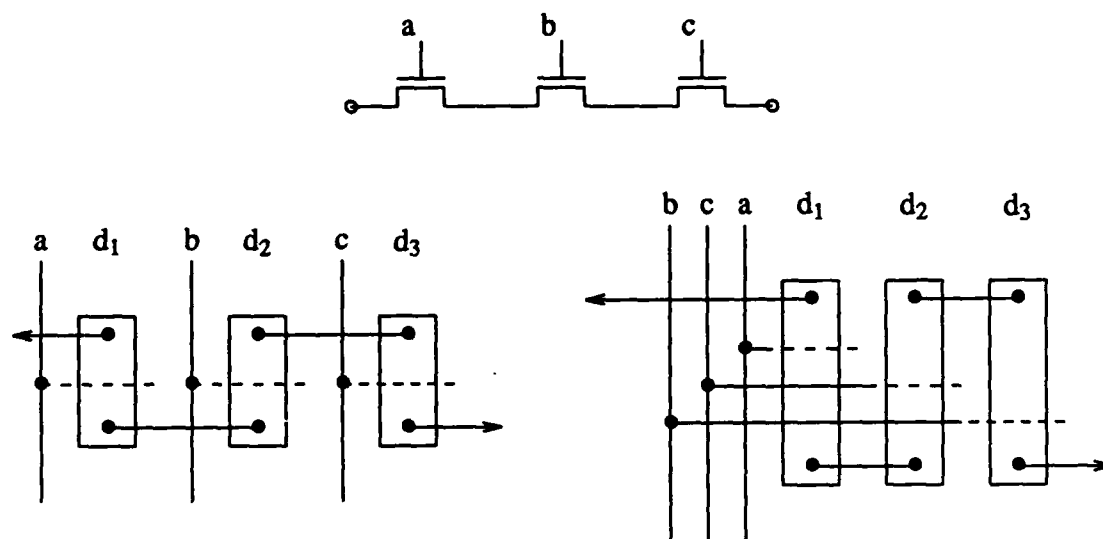


Figure 3.7. Effect of Column Ordering in Metal-Metal Matrix Layouts

process is applied to each subset recursively until a complete order is determined. However, this algorithm does not produce satisfactory results. This approach for determining the column ordering corresponds to minimizing the net-density between columns in the physical layout. However, by minimizing only the net-density between columns, the nets which terminate just before the cut-line are not counted in the partitioning algorithm. Therefore, this net-density does not accurately reflect the number of tracks needed to lay out the circuit, as shown in Figure 3.8.

For example, consider the situation in Figure 3.9(a). The net density between columns B and Y can be improved by exchanging columns A and Y or exchanging columns A and Z. Both exchanges decrease the net density by one, and so they are equivalent. Thus, one of these exchanges would be chosen at random. But both

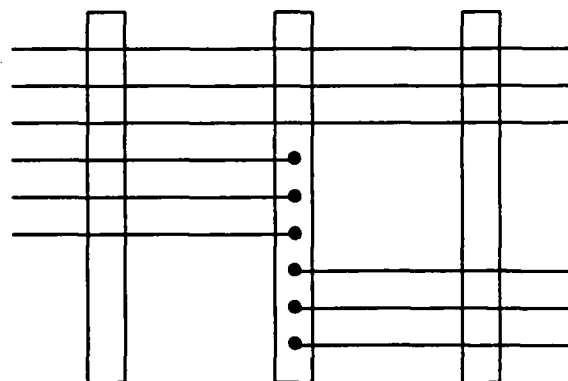


Figure 3.8. Net-Density Between Columns is a Poor Estimate of Track Requirements

exchanges are not equivalent in the physical layout. Figures 3.9(b) and 3.9(c) demonstrate the result of accepting exchange A-Y and exchange A-Z, respectively. Clearly, the ordering in Figure 3.9(c) is better, since the physical layout would require only two tracks instead of three. Thus, an algorithm based on the min-net-cut heuristic between columns would choose the wrong exchange 50 percent of the time in this situation.

3.2.3.1. Min-net-cut across columns algorithm

In the following, an algorithm is presented for minimizing the net-density vertically across a column [Gee89b], rather than between columns. This algorithm has been used recursively to determine a signal ordering that results in smaller metal-metal-matrix layouts than can be obtained by minimizing the net-density between columns.

First, represent each column in the layout by a vertex and each horizontal wire by a multi-terminal net. Divide the vertex set into three sets, A, B, and a set consisting of a

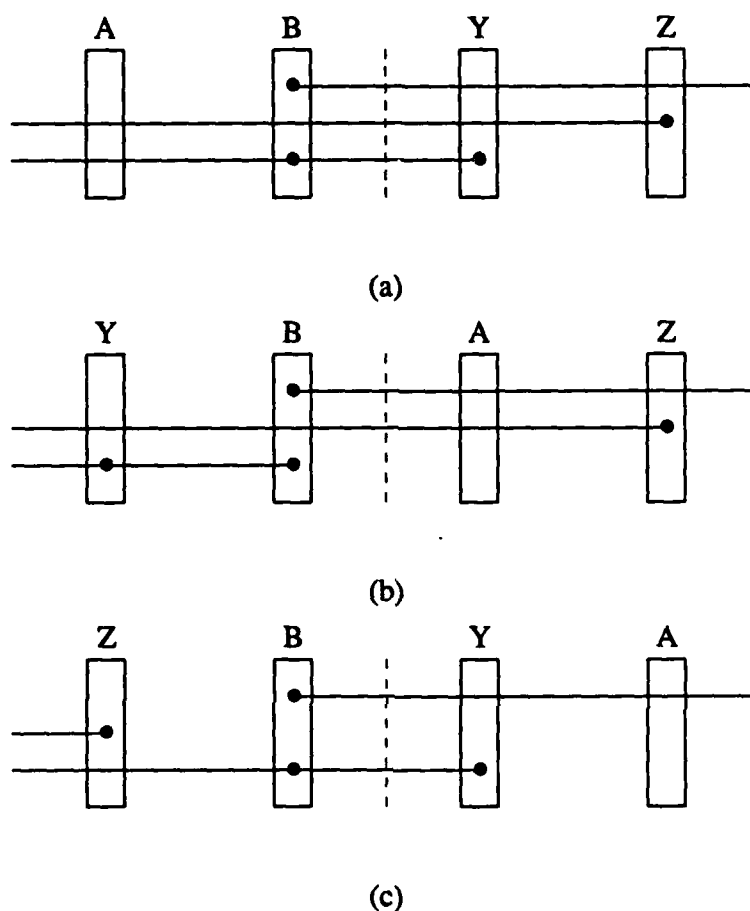


Figure 3.9. Example of Min-Net-Cut Exchanges

(a) Initial Column Ordering

(b) Effect of Exchanging Columns A and Y

(c) Effect of Exchanging Columns A and Z

single vertex, v_c . A discussion on how to choose v_c will be presented later. The initial partitions can be chosen randomly.

Notice that the number of nets which contain vertex v_c plus the number of nets which contain terminals in both sets A and B that do not contain vertex v_c , is equal to the net density of the column represented by v_c . Since the number of nets incident to column

v_c is independent of the partitioning, the net density across this column is minimized by minimizing the number of nets with terminals in sets A and B. Thus, the partition which minimizes the net-density across column v_c is the same as the partition which minimizes the net-density between the sets A and B. Thus, vertex v_c and all the nets which are incident to v_c can be deleted, and then any of the well-known min-net-cut algorithms, such as Kernighan and Lin can be applied to sets A and B [Kern70a].

In order to apply this algorithm to the layout problem, an appropriate v_c must be chosen. One heuristic method would be to try every vertex and choose the partition with the minimum net density. However, this approach would require $O(n)$ more time than the conventional min-net-cut algorithm between columns, and thus the total running time of the algorithm increases to $O(n^3)$.

Two heuristic algorithms, which do not increase the order of the algorithm, have been used to select the cut vertex. First, choose k vertices with the smallest number of nets incident to each vertex, where k is some predetermined constant. Since the objective is to find a partition which minimizes the net-density across a column, if the algorithm starts with a column with a smaller lower bound, the final result may be smaller. The running time of this heuristic is still $O(n^2)$ since k is a fixed constant. Second, choose k vertices with the largest number of nets incident to each vertex. This heuristic attempts to minimize the nets across the *hardest* columns first while the algorithm has more leverage in optimizing the layout, since the lower bound on the number of tracks necessary to lay out the circuit is bounded by the maximum net density of all columns.

The column ordering algorithm chosen in this thesis is a combination of three heuristic algorithms, namely, the conventional min-net cut algorithm between columns and the two previously discussed heuristic algorithms. All three algorithms are applied to the same circuit and the best layout is chosen.

This algorithm has been applied to a number of circuits that were implemented in the metal-metal-matrix layout methodology. The graph in Figure 3.10 compares the effectiveness of both the min-net heuristic and the max-net heuristic as a function of the number of cut vertices chosen. This graph shows that the max-net heuristic obtains the minimal number of rows using just one candidate cut-vertex for most circuits. For both heuristics, all circuits obtained the minimal number of rows by using at most four candi-

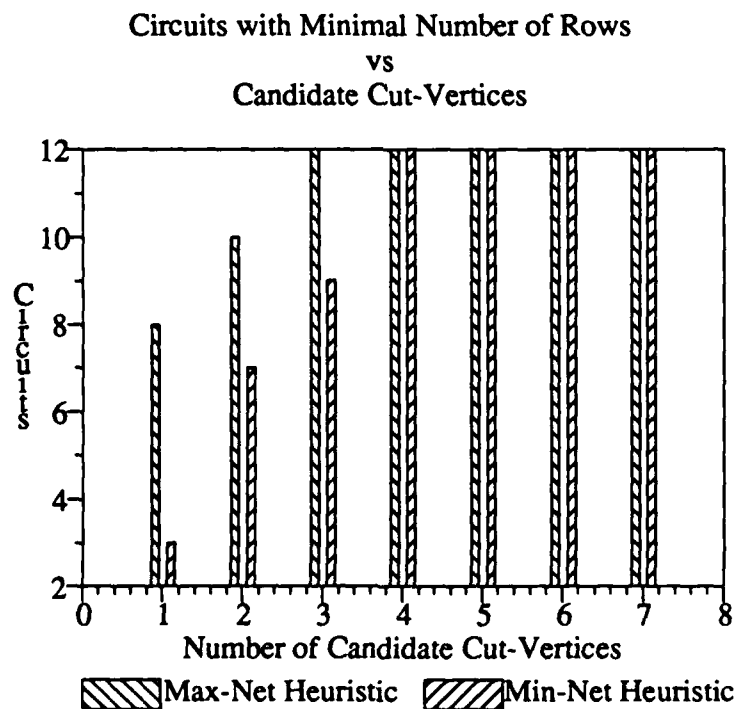


Figure 3.10. Effectiveness of Heuristics

date cut-vertices. Since the run-time increases with the number of candidate cut-vertices, using the three best candidate cut vertices should be sufficient for most circuits.

A comparison between the new approach using both heuristics with the conventional min-net-cut approach between columns is summarized in Table 3.1. The percentage values given in the table represent the improvement compared with the traditional

Table 3.1. A Comparison of Improved Min-Net-Cut Algorithm with Conventional Min-Net-Cut for Metal-Metal-Matrix Layouts

CMOS Circuit	Transistors	Conventional Algorithm (Rows)	Min-net Algorithm (Rows)	Max-net Algorithm (Rows)
ALU4 (74181)	258	86	75 (18%)	75 (18%)
BCD (74145)	96	44	48 (-9%)	45 (-2%)
CMP4 (7485)	140	65	68 (-5%)	60 (8%)
COUNT4 (74163)	270	67	61 (12%)	72 (-4%)
CSA (74183)	42	30	30 (0%)	32 (-7%)
ENCODE (74147)	104	53	53 (0%)	44 (17%)
F. ADDER (7482)	66	31	28 (10%)	29 (6%)
PARITY (74180)	60	28	23 (18%)	23 (18%)
MULT2	50	27	28 (-4%)	30 (-11%)
Comb. Logic	30	18	14 (22%)	14 (22%)
Comb. Logic	16	15	15 (0%)	14 (7%)
Finite State	118	52	52 (0%)	52 (0%)

approach. For both heuristics, the number of cut vertices was limited to the three most promising candidates. From Table 3.1, all three methods are approximately equivalent with respect to the total range of circuits tested. However, if both heuristics are used, then better results are obtained in all circuits, except for two, which show no improvement, and two, which were worse by only one row. By using both heuristics together, the layout has been improved by up to 22 percent with an average improvement of 9 percent over the conventional min-net-cut algorithm. Since both heuristics have execution times of $O(n^2)$, using both heuristics together results in a $O(2n^2) = O(n^2)$ algorithm.

3.2.3.2. CMOS column ordering

The previous algorithm is sufficient for NMOS or dynamic circuits, where the spans of the diffusion columns are equivalent to the spans of the signal columns. However, this is not the case for static CMOS circuits; the span of the diffusion columns in either the P-region or the N-region is only half the span of a signal column. Moreover, any diffusion column in the P-region can be paired with any diffusion column in the N-region, since all PMOS transistors are laid out above the NMOS transistors. Optimizing the P-region separately from the N-region is unsatisfactory, since the signal order in each region would be different in such cases, thus requiring some channel routing to interconnect the two regions.

The algorithm presented in the previous section for optimizing the column order can be applied to CMOS circuits if the min-net-cut algorithm is modified to account for the extra degree of freedom in CMOS circuits [Gee89c]. This freedom in CMOS circuits can be modeled by representing the columns with three classes of vertices: signals (S), P-

diffusion (P), and N-diffusion (N). An initial partition can be improved by considering five different exchanges, namely, three intra-class exchanges (S-S, P-P, N-N) and two inter-class exchanges (S-PN, PN-S). A brief sketch of the modified min-net-cut algorithm is shown in Figure 3.11. The run time of this algorithm is still $O(n^2)$ since a constant amount of time is spent in Step 3 to determine the best exchange.

As an example, Figure 3.13(a) represents the nets symbolically before the columns are ordered for the compacted incidence matrix in Figures 3.12(a) and (b). Columns 1n through 7n correspond to the N nodes which are the 7 columns in the compacted incidence matrix for the NMOS region and columns 1p through 7p correspond to the P nodes. After applying this procedure to Figure 3.13(a) the columns are reordered as

1. For all nodes compute the change in net density if only that node is moved to the other side.
2. Choose $3n$ nodes (n signals, n P-diffusion, and n N-diffusion) which have the largest change in net density as candidates to be exchanged. Empirically, $n = 2$ or 3 produces good results.
3. From the candidate nodes chosen in Step 2, choose the exchange type which maximizes the gain:

exchange type	gain
S-S	$Da[u] + Db[v] - Cnet(u,v)$
P-P	$Da[u] + Db[v] - Cnet(u,v)$
N-N	$Da[u] + Db[v] - Cnet(u,v)$
S-PN	$Da[u] + Db[v] + Db[w] - Cnet(u,v) - Cnet(u,w)$
PN-S	$Da[u] + Da[w] + Db[v] - Cnet(u,v) - Cnet(w,v)$

where $Cnet$ is a correction term for nets which contain both nodes.

4. If the set is not empty then go to Step 2.
5. Choose the first k exchanges to be the exchange set, where

$$\text{total gain} = \max_k \left(\sum_{i=0}^k \text{gain}[i] \right).$$
6. If total gain is positive then go to Step 1.

Figure 3.11. CMOS Ordering Algorithm

							B	A	B	V	A	E	W							
							C	D	C											
									D	U										
U	V	D	U	B	C	D	A	A												
		C					E	B												
								W	V											
		1	1	0	0	0	0	1												
		2	0	0	0	0	0	2												
		0	0	0	0	0	0	2												
		0	0	1	1	1	0	0												
		0	0	0	0	0	1	0												
V	U	B	A	B	V	A	E	W												
		C	D	C																
		D	U																	
		1	1	1	1	1	1	1												
		0	0	0	0	1	1	1												
		2	0	0	0	0	0	0												
		2	0	0	0	0	0	0												
		1	0	0	0	0	0	0												
F	U	0	2	0	0	0	0	0												
		0	2	0	0	0	0	0												
		0	2	0	0	0	0	0												
		0	0	2	0	0	0	0												
		0	0	2	0	0	0	0												
		0	1	1	1	0	0	0												

(a)

(b)

Figure 3.12. CMOS Compacted Incidence Matrix
 (a) PMOS Region
 (b) NMOS Region

shown in Figure 3.13(b). A comparison of the Kernighan-Lin min-net-cut algorithm with and without the CMOS extensions is shown in Table 3.2. From the table we see that the new algorithm has reduced the number of rows on average by 17 percent.

3.2.3.3. Constraint detection

The height of the subcircuit can be bounded from below by two quantities, namely, the maximum net density over all columns and the length of the longest path in the

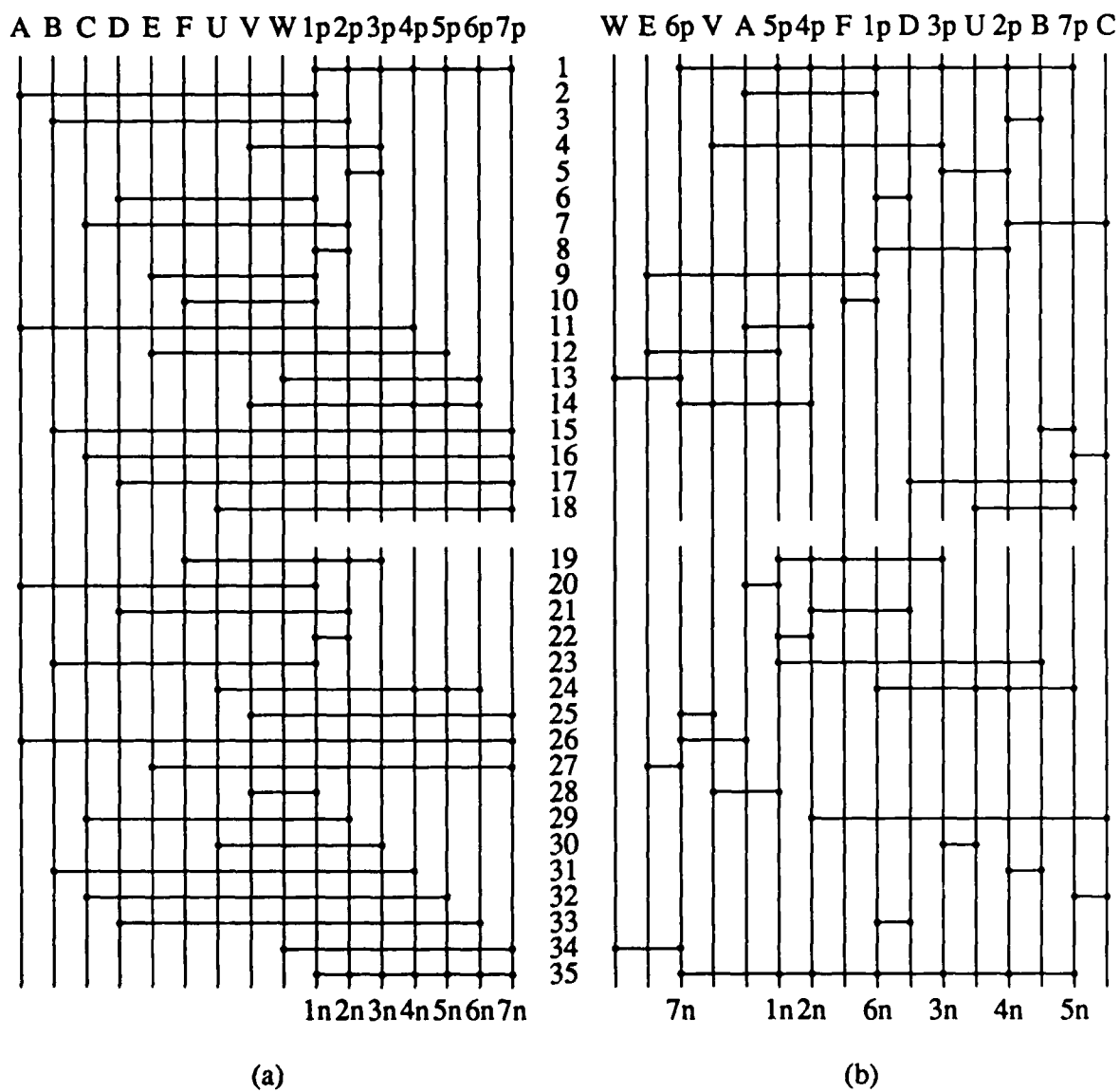


Figure 3.13. Net Graph Before and After Column Ordering
 (a) Before Column Ordering
 (b) After Column Ordering

Table 3.2. A Comparison of CMOS Ordering Algorithm with K-L Min-Net-Cut for Metal-Metal-Matrix Layouts

CMOS Circuit	Transistors	K-L Algorithm (Rows)	CMOS Algorithm (Rows)	Percent Improvement
ALU4 (74181)	258	112	86	(33%)
BCD (74145)	96	64	44	(31%)
CMP4 (7485)	140	71	65	(8%)
COUNT4 (74163)	270	98	67	(32%)
CSA (74183)	42	33	30	(9%)
ENCODE (74147)	104	64	53	(17%)
F. ADDER (7482)	36	31	24	(23%)
PARITY (74180)	60	34	28	(18%)
MULT2	50	31	27	(13%)
Comb. Logic	30	18	18	(0%)
Comb. Logic	16	15	15	(0%)
Finite State	118	66	52	(21%)

corresponding vertical constraint graph. The vertical constraint graph is an acyclic directed graph representing the vertical constraints between the horizontal nets. The maximum net density is minimized by reordering the columns. The length of the longest

path in the constraint graph can be reduced by removing vertical constraints in the path. This can be achieved by adding additional columns to the layout.

The algorithm to determine which minimal set of constraints to remove is based on a breadth-first traversal of the vertical constraint graph [Gee89d]. The range of possible heights versus number of constraints removed for each vertex is computed incrementally at each level from the information saved at its parents.

A lower bound on the height of the cell can be determined from the corresponding constraint graph $G(V, \vec{E})$. The vertices in this graph represent source-drain nets and the edges represent constraints between these nets. The edges are given two attributes, a weight equal to the number of transistors between the two corresponding nets and a column number corresponding to its column in the compacted incidence matrix. Two additional vertices are added to this graph, namely, a source vertex and a sink vertex. As their names imply, the source vertex is connected to each parentless vertex and the sink vertex is connected to each childless vertex. The weight of these edges is zero. Figure 3.14, shows a compacted incidence matrix and its constraint graph. A lower bound on the height of the cells is given by the heaviest path between the source and the sink vertices, where the weight of a path is given by the sum of its edge and vertex weights. All vertices have weight one, except for the source and sink vertices which have weight zero. For example, the minimum height for the incidence matrix in Figure 3.14 is 9. Inter-column constraints can be removed by adding an additional column to the layout as shown in Figure 3.15. This corresponds to splitting a vertex into several vertices, as shown in Figure 3.16, which reduces the height of this cell to 7. The problem is to deter-

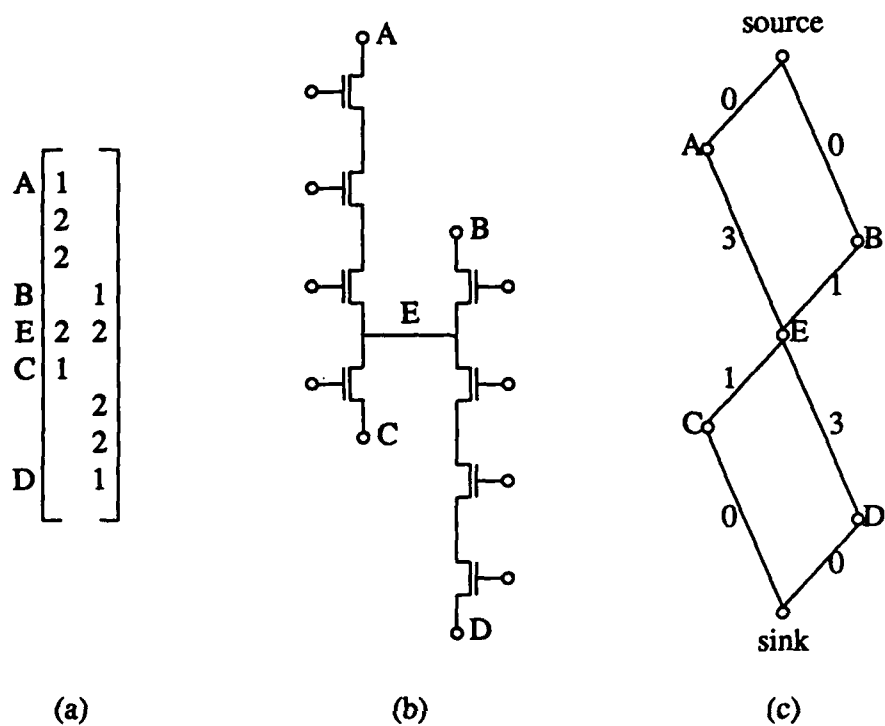


Figure 3.14. Compacted Incidence Matrix and its Constraint Graph

- (a) Transistor Circuit
 (b) Incidence Matrix
 (c) Vertical Constraint Graph

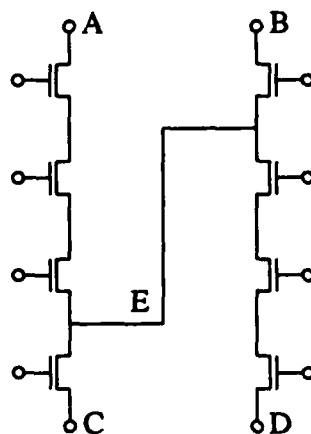


Figure 3.15. Horizontal Constraints From Net E in Figure 3.14 can be Removed by Adding an Additional Column to the Layout

mine which source-drain net constraints to break that will require the fewest additional columns, while achieving the specified height objective.

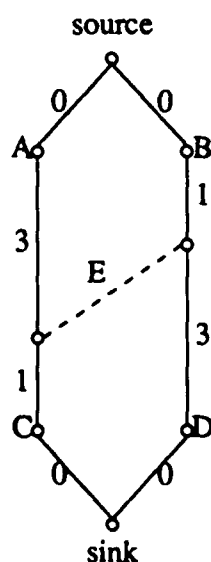


Figure 3.16. Effect of Removing the Vertical Constraints from Net E on the Constraint Graph

This can be computed efficiently if the vertices in the constraint graph are ordered. To determine the order, assign each vertex a level equal to its maximum distance from the source vertex, where distance is measured in terms of the number of edges. The vertices in the graph are then processed level by level starting with the vertices with the lowest level. So for each vertex at level l , the range of possible heights for vertex v versus the number of constraints removed is computed. This can be computed efficiently from the vertices at level $l-1$. The minimum height of vertex v if n_c constraints are removed is

$$\text{minheight}(v, \text{col}, n_c) = \min_{(v,u) \in E} (\text{minheight}(u, \text{col}, n_c - 1) + \text{wgt}(v, u)),$$

$$\max_{\sum_{(v,u) \in E} x_u = n_c} (\text{minheight}(u, *, x_u) + \text{wgt}(v, u)))$$

where $\text{minheight}(v, *, n_c)$ is the maximum value over all columns which are incident to vertex v . Now the minimum height of the constraint graph is given by

$$\text{minimum constraint graph height} = \text{minheight}(\text{sink}, *, n_c)$$

with a corresponding cell width equal to $n_c + C_0$ columns, where C_0 is the width of the cell without any constraints removed. The actual constraints to remove can easily be determined by saving two arrays of values at each vertex, one representing the minimum height of the vertex for n_c constraints removed and the other stating whether or not the constraint represented by the current vertex should be removed given that n_c constraints are removed.

Although the worst case behavior of this algorithm is $O(2^n)$ where n is the number of vertices, the running time is bounded by $O(n^w)$, where w is the maximum number of transistors connected in parallel.

3.2.3.4. Track assignment

Each incidence matrix defines two sets of nets. The first set is represented explicitly by each row of the matrix and corresponds to interconnections between the sources and drains of different transistors. The second set is represented implicitly by the transistor labeling in each column and corresponds to nets connecting the signal columns to the gates of the transistors. These nets and their column ordering are now transformed into a symbolic layout. Certain constraints force the track assignments to follow a certain order since the gate of a MOSFET must be assigned between the source and drain. A vertical constraint graph for the NMOS portion of the circuit is generated from the column-compacted matrix in Figure 3.12 and is shown in Figure 3.17. Notice that NET 7 and 8 are at the same level. This reflects the fact that these nets represent series transistors and that their order can be permuted. For this group of series transistors, an extra node is added in series, so that the length of the longest path is a lower bound on the number of tracks required. In general, $n-1$ nodes must be added in series for each group of n series transistors.

The symbolic layout is now generated one track at a time. All nets which have all their constraints satisfied are candidates to be assigned to the first track. However, not all candidates are assigned with equal priority. The net priority is determined by its distance from the root node in the vertical constraint graph, since this distance represents the

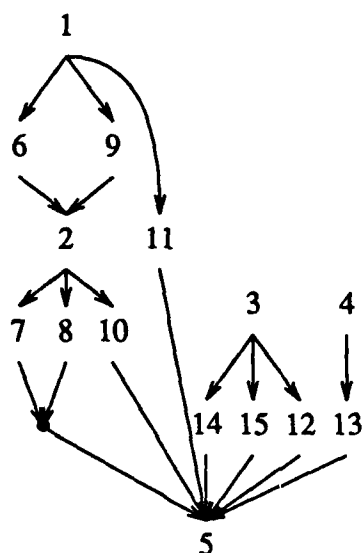


Figure 3.17. Vertical Constraint Graph

minimum number of tracks still necessary to complete the layout. For nets with equal priority, the nets with the left-most terminal is assigned first. When the current track is full, a new track is allocated and the process is repeated until all nets have been assigned. Applying the above procedure to the compacted incidence matrix in Figure 3.12 results in a symbolic layout shown in Figure 3.18.

3.3. Summary

This chapter has presented a symbolic representation for MOS circuits, namely, the incidence matrix representation. In general, the incidence matrix is very sparse. To achieve a compact layout, the sparsity of the incidence matrix is first reduced by merging columns and the number of horizontal tracks is reduced by reordering the merged columns. The column ordering algorithm minimizes the net-density across the columns,

```

WE VA  F D U B C
| | T + + * * + * + * + T | | | | | | | | | |
| | | | | | | P T | | | | |
| | | | T + + + p T + + + + p |
| | | T + + + + + + p | P T | |
| T + + + p | | P + + T P + P T
T + p | T + p T + + + + T T p |
| | T * + * T T * + T T - + T | | | | | | | | |
| | | | | | | | | | |
| | T T | T * * - + T T * + T |
T + n | T n | | T + N T N T N T
| T n | | | N + N T | | | | |
| | N + T T T | | | | | | |
| | | T + r N + + + + + + + T
| | | | N + + + + + + + T | |
| | T + + * * + * + * + * + T |

```

T Two layers electrically connected and form a 'T'
 N NMOS transistor with gate signal to the right
 n NMOS transistor with gate signal to the left
 P PMOS transistor with gate signal to the right
 p PMOS transistor with gate signal to the left
 + two layers cross but are not electrically connected
 * two layers cross and are electrically connected
 | vertical metal (top layer) or diffusion
 - horizontal metal (bottom layer)

Figure 3.18. Symbolic Layout

which is a more accurate lower bound on the number of tracks than the net-density between columns. Thus, denser circuits are generated. The column ordering algorithm optimizes both PMOS and NMOS transistor regions simultaneously to minimize the cell area. The compacted incidence matrix is then converted into a symbolic layout. This symbolic layout can be transformed into the mask data necessary for fabrication using the methods to be discussed in the next chapter.

CHAPTER 4.

SYMBOLIC LAYOUT

4.1. Introduction

In VLSI circuit design, there are two main computer-aided tools available for developing custom cells, namely, graphics layout and symbolic layout. In the graphics layout method, a designer must work with the geometric design rules. To assist the designer, graphics editors, such as MAGIC [Oust84a, Tayl84a], have incorporated *plowing* (a form of compaction) and interactive design rule checking (DRC). However, different topological instances of a circuit are still difficult and very time-consuming to generate. The circuit is very technologically dependent, rendering the circuit obsolete with advancing fabrication technology.

The symbolic layout methodology, on the other hand, uses different symbols to represent different geometric primitives. This simplifies the layout process by hiding the complexity of the geometric constraints. This is advantageous to both the designer and the CAD tools, such as design rule checkers, circuit extractors, and logic verifiers, which no longer need to work with geometric objects. An additional benefit from this level of abstraction is that the symbolic layouts are process independent. A number of symbolic layout systems have been developed in the past for MOS cells, such as SLIC [Gibs78a], STICKS [Will78a], SLIM [Dunl80a], MULGA [West81a], SLS [Posl85a], SYMPLE [Szab87a], and MGX [Tera87a]. Most of the above systems require a graphics

terminal/plotter to edit/display the symbolic layouts. In addition, the symbols and/or wire segments are associated with specific layers.

This chapter presents a symbolic layout system [Gee88a] for single-poly, double-metal MOS technology. The symbolic layout system uses a variable grid-base approach and lays out circuits using the M^3 layout methodology. Unlike other symbolic layout systems, this system provides an additional level of abstraction. The layout symbols represent simple geometric primitives. The actual mask layers used to construct these primitives are determined automatically. This allows one symbolic layout to represent both M^3 layout variations [Gee89a, Kang87a]. Section 4.2 begins with an overview of the symbolic layout system. Next, the layout primitives used by the symbolic layout system will be discussed as well as the techniques employed to minimize the area. Finally, this chapter concludes with a comparison of the layout area using this approach with the standard cell approach.

4.2. Symbolic layout system overview

This symbolic layout system has been developed to lay out MOS circuits in the M^3 methodology. Dense layouts for different MOS technologies can be produced by simply specifying a new technology-file. A technology-file for a typical 2-micron process is shown in Figure 4.1. The technology-file specifies the minimum layer-to-layer spacing, minimum layer widths, and other technologically specific features. Currently, mask data for 3-micron single-tub technology and 2-micron twin-tub technology can be generated.

The M^3 layout style has an orderly structure which easily maps into a variable 2-dimensional grid. By restricting the symbolic layout elements to lie on grid points, the

```

#   Design Rules in tenths of Lambda.
#   Lambda in centimicrons.
#
version "2.0 micron rules
#       LAYER SPACINGS
#       CC   CD   CS   CM   CM2  CP   CNW  CW   CV
contact    20   20   20   20     0   20    0    0   40
diffusion  20   30   30    0     0   10   40   40    0
pdiffusion 20   30   30    0     0   10   40   40    0
metal      20    0    0   30     0    0    0    0   20
metal2     0    0    0    0    40    0    0    0   20
poly       20   10   10    0     0   20    0    0   20
nwell      0   40   40    0     0    0   30   30    0
pwell      0   40   40    0     0    0   30   30    0
via        40    0    0   20    20   20    0    0   20

# LAYER WIDTHS
width      20   30   30   20    30   20   110  110   20

# NAIL HEAD SIZES
intersect  *   40   40   40    40   40    *    *    *

# layer numbers for extraction labeling
layernumbers  4    1    9    5     7    3    11    2    6

# overlap (diffusion overlap for trans) (poly overlap for trans)
overlap      20   20

# lambda = 1 micron
lambda       10

# WELL REQUIREMENTS
wantnwell    1   # need NWELL
wantpwell    1   # need PWELL

# TIMBERWOLF scale factor
timberwolf   5

```

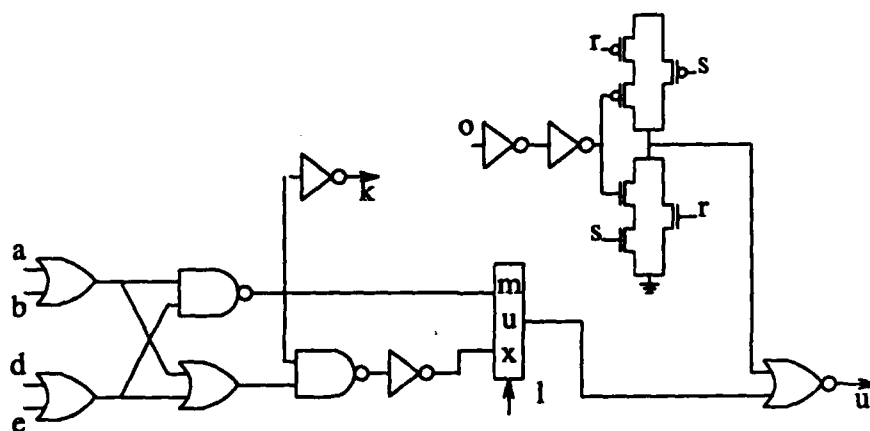
Figure 4.1. Technology-File for 2-Micron CMOS Twin-Tub Process

symbolic layout can be stored efficiently as a 2-dimensional array within the computer. In addition, the symbolic layout elements can be represented as ASCII-characters and displayed on any ASCII terminal or line printer. Thus, the symbolic layout can be easily manipulated by a human designer or other design automation tools. The grid spacing used to place the layout symbols is neither fixed nor uniform. This allows layouts to have a high packing density even though the design rules are different for different mask layers. The circuit in Figure 4.2 was laid out using this symbolic layout system.

4.3. Layout symbols and symbol mapping

The system consists of a small set of symbolic layout elements, shown in Table 4.1. These symbols represent layer independent and orientation independent geometric primitives. Each symbol is interpreted with respect to its adjacent symbols and its functionality to determine the proper layers and the orientation of the symbols. For instance the symbol '-' represents a horizontal wire segment. In the M^3 layout style, this wire segment can be metal, poly, or both. The neighboring symbols are inspected to classify the wire segment based on length and functionality. For short wire segments interconnecting the gates of transistors the wire will be laid out in poly provided that no extraneous transistors are created. Otherwise, the wire will be laid out in metal and appropriate layer changing contacts will be added where necessary.

Internally, each layout symbol is converted into bit flags representing each layer. Each layer is represented by 4 bits denoting all four directions (up, down, right, and left). Thus, in this current implementation, 8 routing layers can be represented in one 32-bit word, of which only 4 are used (diffusion, poly, metal 1, and metal 2). This



(a)

```

ab duse rt qmcf lhn p ogj ik
||T+++T || |T+- -++T| ||| ||
||PTT++T||TT|||T-+TTpTT||| ||
||T+++++++*pT++TT++++TT+T
||P+++TP++++TT+pPT| |P+++++T|
||TT|||P+TP+++++++TPTT+p||
T+T|||T++*++++*++++*++T||
|Tp|||PT| |Tp| ||| |||
||T+++++++*++p|T+p| |||
T+p||T+pT+p|||T+++++ -*+p||
|TT|||T+*T|T+- -T||T+- -++*T| | | | | | | | | |
|| ||| ||| ||| ||| |||
|| |T++T|| |T+- -++*+-++T||
||T++++N+- -T|| T++*nTT|||
||N++++*+*T||| |T+ ++++++n|
||T+TT+nT+nT++TNT| |T++TTT|
|| |||N++T|||N++++TNT| |NT|
||T++++*+*++++*++++*++T||
T+nT+++n||N++TTnn|T+++n|T+n|
|Tn|||Tn||T+++++*+T|||N+T|T+T
||T+++++ -++*+-++n|T+++ -T|
|| |||T+- -++TT-++T| ||| ||

```

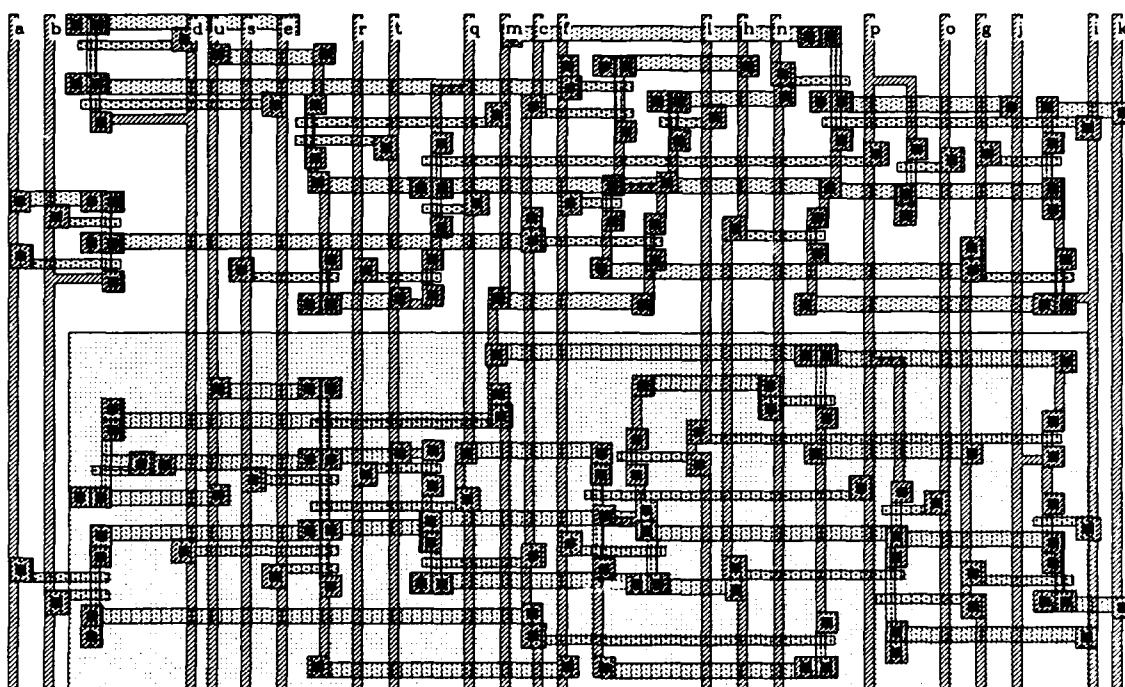
(b)

Figure 4.2. Circuit Schematic, Symbolic Layout, and Mask Data

(a) Circuit Schematic

(b) Symbolic Layout

(Continued on next page)

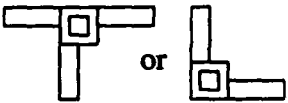

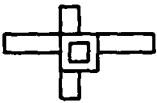




(c)

Figure 4.2 (Continued)

(c) Mask Data

Table 4.1. Layout Symbols

Symbol	Geometric Primitive	Symbol	Geometric Primitive
T			
*		—	
n,N	Nmos Transistors	+	
p,P	Pmos Transistors	space	No Geometry

representation allows the symbolic layout generator to simultaneously choose all the layers between two adjacent symbols consistently by using bit-shift and bitwise-and operations. Each layout symbol is mapped into one general bit-flag representation, and then, it is made consistent with its adjacent symbols in another pass. For instance the symbol '-' is initially mapped into both poly and metal 1 going both right and left. After the internal representation of the symbolic layout is consistent, the number of vias is reduced by maximizing the use of metal 1 runners where possible.

4.4. Grid spacing

Each layout symbol is associated with a set of attributes which describe the relevant physical information for each symbol, such as, wire widths or transistor length and width. This information, along with the design rules, and the contact orientation for multi-layer transitions (diffusion to metal 1 to metal 2, or poly to metal 1 to metal 2) will determine the grid spacing. Of these factors, only the contact orientation can be varied by iSILVER to minimize the width of its corresponding row or column.

The algorithm to determine the proper orientation for each contact begins by eliminating all contacts which can overlap an adjacent symbol. The orientation for these contacts is already determined and will not impact the row or column spacing. Next, all the rows with column prime contacts and all the columns with row prime contacts are removed from consideration, as shown in Figure 4.3(a). A column (row) *prime* contact is defined as any contact which is in a column (row) by itself. These contacts are oriented to minimize the length of their respective rows or columns. This step is repeated until all prime contacts have been eliminated. At this point, the row or column with the most contacts is chosen, Figure 4.3(b). The orientation of each of these contacts is chosen to minimize the length of the row or column, respectively. This row or column is then eliminated from further consideration and the process is repeated until all contacts have been assigned an orientation, Figure 4.3(c).

Unlike other symbolic layout systems which employ some form of compaction, such as constraint-graph or critical path, super compaction, or zone refinement, the symbolic layout system produces dense layouts by varying the spacing between grid lines.

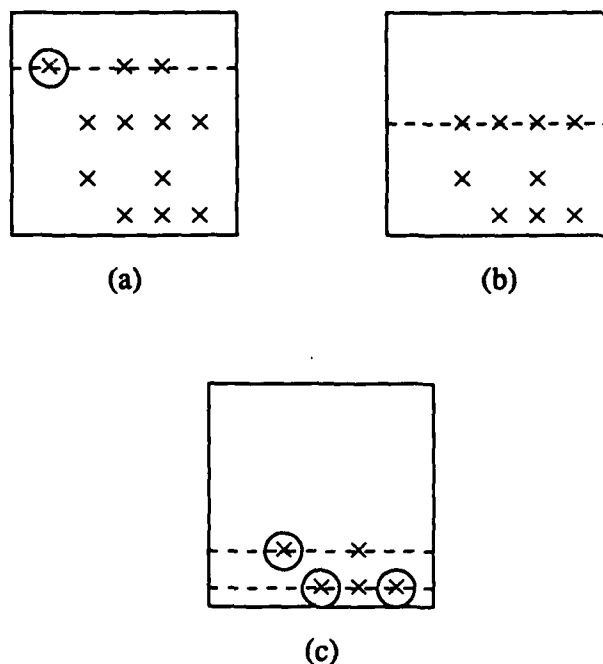


Figure 4.3. Progression of Contact Orientation Algorithm

- (a) Prime Contact is Circled. First Row Contains a Column Prime Contact.
 (b) The Second Row With the Most Contacts is Eliminated.
 (c) Third and Fourth Rows Contain Column Prime Contacts.

The minimum spacing between grid lines can easily be computed by inspecting the symbols in adjacent rows or columns. For general layouts, this approach may produce layouts which are very area inefficient compared to any of the above compaction approaches, since every layout element in a row or column must be moved together. However, this is not the case for M^3 layouts. The M^3 layouts consists of grid-like structures, namely, signal columns and horizontal routing tracks. These structures are nearly uniform in width, which is ideal for the variable grid approach. This approach is also much faster than any of the above compaction schemes. An important feature of the

variable grid approach is that cells can be generated with a specified signal pitch to simplify routing or to allow it to be connected to other cells by abutment.

4.5. Results and comparisons

For comparison, four circuits have been laid out in different layout styles. Table 4.2 below provides a layout comparison of PLA, Gate Matrix [Lope80a], the double-metal version of Gate Matrix, and the M^3 in 3-micron design rules. The PLAs were simply-folded and multiply-folded using PLEASURE [Hach82a] and laid out using PANDA [Mah84a] for the single-metal PLAs and GENESIS [Cowa87a] for the double-metal PLAs. Circuits 1 and 2 are combinational logic and Circuits 3 and 4 are finite state machines. Circuit 1 has 29 transistors in PLA implementation with two-level AND-OR-gate design and 15 transistors in Gate Matrices and M^3 with complex cell design. Circuit 2 has 30 transistors in PLA and 12 transistors in Gate Matrices and M^3 , respectively. Circuit 3 has 76 transistors in PLA and 61 transistors in Gate Matrix and M^3 , respectively. Circuit 4 is the first example in Wing [Wing85a] which has 118 transistors. The

Table 4.2. Layout Area for PLA, Gate Matrix, and M^3

Circuit	Single-metal		Double-metal		
	PLA (μm^2)	Gate Matrix (μm^2)	PLA (μm^2)	Gate Matrix† (μm^2)	M^3 (μm^2)
1	10152	7680	17113	9480	8640
2	21648	8448	27475	9344	8567
3	123480	103488	218010	108192	102818
4	-	185856	-	195712	200322

† The double-metal version of gate matrix uses the top-level metal lines on top of the poly gate lines.

results in Table 4.2 show the smallest area of PLAs produced by the tools used. The PLA area in Table 4.2 is the area of the core only. A comparison between a standard cell approach and the M^3 layout style is shown in Table 4.3. The area of the standard cells consists only of the area of the cells plus the channel area for inter-cell wiring. In order to make a fair comparison with standard cells, the transistor widths in the M^3 layouts are the same as the transistor widths in the standard cells.

The gate matrix layouts were hand-generated for circuits 1, 2, and 3. An initial signal order was determined by using a min-net-cut procedure. Then the nets were assigned to tracks allowing series transistors to be permuted. Finally, additional improvements to the gate matrix were sought by intelligently permuting the signal order. Circuit 4 is the first example in Wing [Wing85a]. The double-metal gate matrix is the same as the single metal gate matrix except that metal2 is connected on top of the poly lines to reduce the parasitic resistance. However, it should be noted that this configuration suffers from the large increase in the parasitic capacitance in the gate signal lines by as much as 60 percent. This simple-minded approach to double-metal gate matrix layout demonstrates that a much higher area-penalty is incurred if the extra metal layer is not used effectively.

Table 4.3. Layout Area for Standard Cells and M^3

Circuit	Standard Cell (μm^2)	M^3 (μm^2)
1	32424	18056
2	30048	17902
3	276192	207267
4	213852	204612

Also, for large layouts, additional poly-metal1-metal2 contacts must be added in the core to achieve a similar reduction in the parasitic resistance as in the M^3 layouts. However, extra contacts are difficult if not impossible to add, in the congested areas of the gate matrix without increasing the area further. The area figures in Table 4.2 for the double-metal gate matrix reflect the overhead of adding only two rows of contacts at the top and bottom of the corresponding single-metal gate matrix circuit.

From Table 4.2, one can see that the layout density of M^3 is much higher than that of PLA implementations. Although the layout area of M^3 is sometimes larger than that of the single-metal gate matrix, its circuit speed is up to 45 percent faster. In general, the relative increase in circuit speed is higher for larger circuits which generally have longer poly runners. In comparison, the double-metal version of the gate matrix layout takes a larger area than M^3 for comparable circuit speed. It should be noted that for Circuit 4 the double-metal gate matrix layout is about 3 percent smaller than the M^3 layout since only two contact rows are used between poly and metal2. However, if more contacts are inserted to reduce the parasitic resistance in the gate signal path comparable to that of M^3 layout, the double-metal gate matrix layout area will become larger.

The cells used in the standard cell approach are from the CMOS3 CELL LIBRARY [Hein88a]. Although these cells have been designed using only a single layer of metal, the availability of an additional layer will not reduce the size of simple cells such as inverters, NAND, NOR, etc. The cells were placed using TimberWolf version 3.3 [Sech86a] and interconnected using two layers of metal with YACR [Reed85a].

Table 4.3 shows that the symbolic layout is usually smaller than the standard cell implementation. However, the symbolic layout is not restricted to the cells available in the standard cell library. The above circuits could have been implemented with complex gates, or switching network logic [Gee89a] could have been applied to reduce the number of transistors. In addition, if these cells are to be used in an actual circuit, the transistors in the symbolic layout would be optimized, which would reduce the size of many transistors, thus the size of the symbolic layouts would be even smaller.

All of the transistors in Table 4.2 were generated using minimum size transistors. In many circuits the transistor widths are increased to reduce the delay or to drive multiple transistors. Table 3 compares the impact of increasing the widths of all transistors for the M^3 circuits in Table 4.2. From Table 4.4, we can see that doubling or quadrupling all the transistors in these circuits results in only a modest increase in circuit size. The reason for this behavior is that many transistors can be extended under the signal columns.

Table 4.4. The Impact of Different Transistor Sizes on Circuit Area

Circuit	Relative Circuit Area			
	Transistor Size			
	x1	x2	x4	x10
1	1	1.014	1.133	1.720
2	1	1.016	1.101	1.610
3	1	1.000	1.132	1.864
4	1	1.019	1.076	1.517

4.6. Summary

In this chapter, a symbolic cell generator for double-metal, single poly MOS technology using the M^3 layout methodology has been presented. The symbols used in this symbolic layout system provide an additional level of abstraction over other symbolic layout generators by letting the layout symbols represent geometric primitives not associated with any specific layers or orientation. This gives the symbolic layout generator greater leverage in optimizing the layout. Also, restricting the layouts to use the M^3 layout methodology enabled the use of simple, yet powerful algorithms for generating dense layouts.

By using logic design tools such as switching network logic [Gee89a] and the techniques presented in Chapter 3 in conjunction with this symbolic layout system, physical layouts can be generated from truth tables for combinational logic or from transition tables for sequential logic. This implementation of the 74181 4-bit ALU with 258 transistors is shown in Figure 4.4 and required 205 seconds to compact the incidence matrix and to lay out.

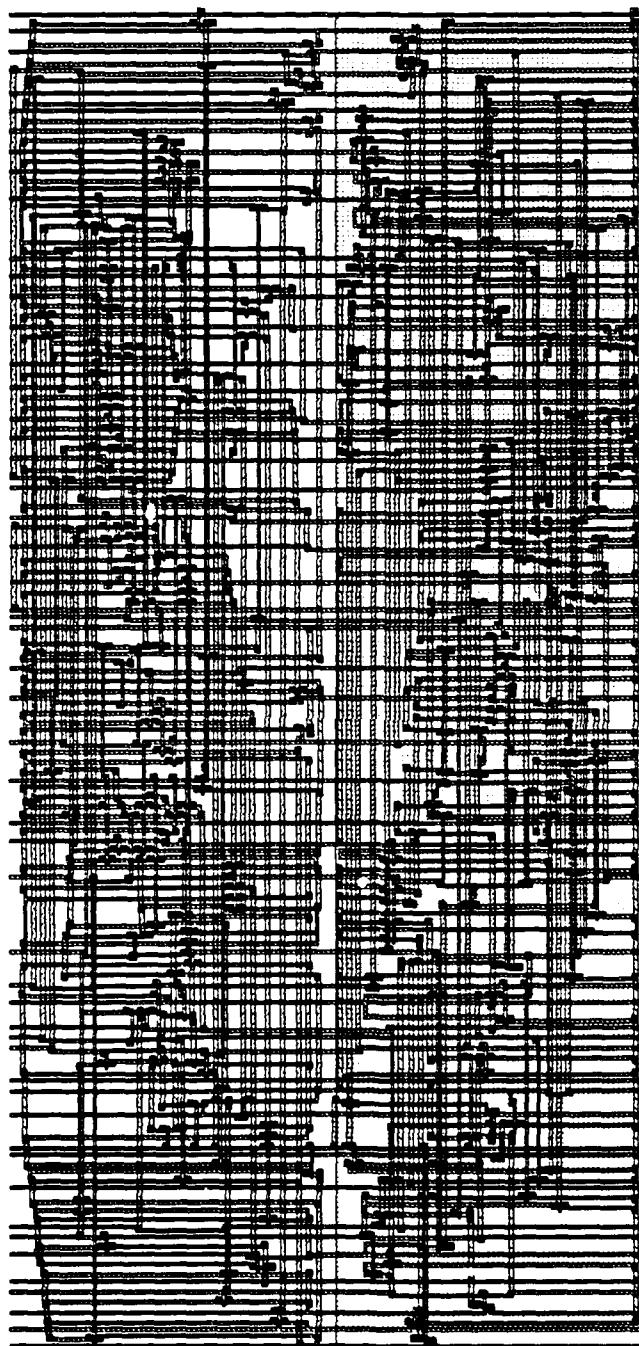


Figure 4.4. An M³ Layout of the 74181 4-Bit ALU

CHAPTER 5.

iSITE: AUTOMATIC MODULE GENERATION SYSTEM

5.1. System overview

A schematic of the module generation system is shown in Figure 5.1. The input to the cell generation system is a transistor netlist. This can be expressed by a SPICE file or by incidence matrices. This transistor netlist is partitioned into channel-connected transistors. Channel-connected transistor groups correspond to conventional gates such as NANDs and NORs, complex-gates, and pass-transistor gates. Although individual gates can be laid out separately as in the conventional standard cell approach, this cell generator lays out larger blocks to obtain a higher packing density. Each channel-connected transistor group is considered as one indivisible unit and a collection of these groups is then merged together based on their connectivity to form cells of approximately 100-200 transistors. Although circuits do exist which have channel-connected transistor groups larger than 200 transistors, such as memory arrays, most random logic circuits consist of small groups. Hence, the discussion is focused on determining good algorithms to merge small groups together instead of splitting large groups. The reason that a channel-connected transistor gate is considered to be an indivisible unit for layout is to avoid the possibility of adding high parasitic capacitance on an internal gate node from intercell routing.

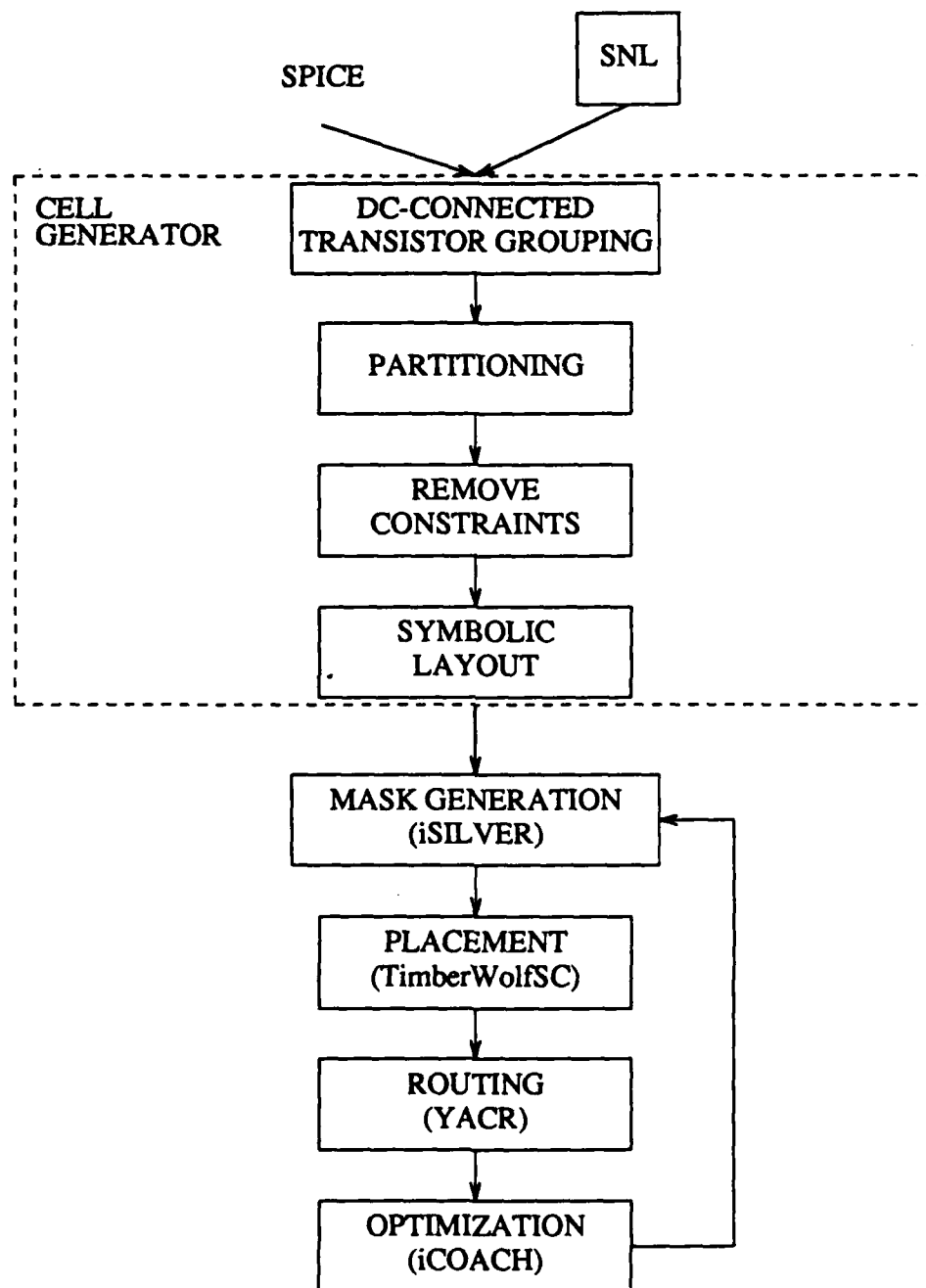


Figure 5.1. iSITE System Overview

A symbolic layout is then generated for each of the subcircuits using the methods presented in Chapter 3. This symbolic layout is converted into the physical mask layers, and the transistors along with the parasitic resistances and capacitances are extracted using iCPEX. This transistor circuit description is converted into a gate-level description and then optimized using iCOACH. The symbolic layout, along with the new transistor sizes, is used to generate a new physical layout. These cells are then placed using TimberWolf and interconnected using YACR.

5.2. Circuit specification

The input specification for the iSITE system can be an incidence matrix or SPICE format. The SPICE format can be converted into an incidence matrix using the method described in Chapter 3. iSITE allows nested subcircuit definitions and has 3 predefined static CMOS gates, namely, INVERTER, XOR, and XNOR. The system also has 6 predefined static CMOS variable size input gates, which are n-input NAND, n-input NOR, n-input AND, n-input OR, AOI and OAI gates. The transistor configuration for these gates is shown in Figure 5.2. A larger library of gates can easily be added to the iSITE system. For instance, the JK FLIP-FLOP in Figure 5.3 could be defined with the following subcircuit definition:

```
.SUBCKT      J      K      CLK  Q      JKFF
X1           K      Q      CLK  1      NAND3
X2           J      QBAR  CLK  2      NAND3
X3           CLK  QBAR  2      QBAR  Q      AOI22
X4           CLK  Q      1      Q      QBAR  AOI22
.ENDS
```

The user-defined and the predefined subcircuits in iSITE are much easier to maintain

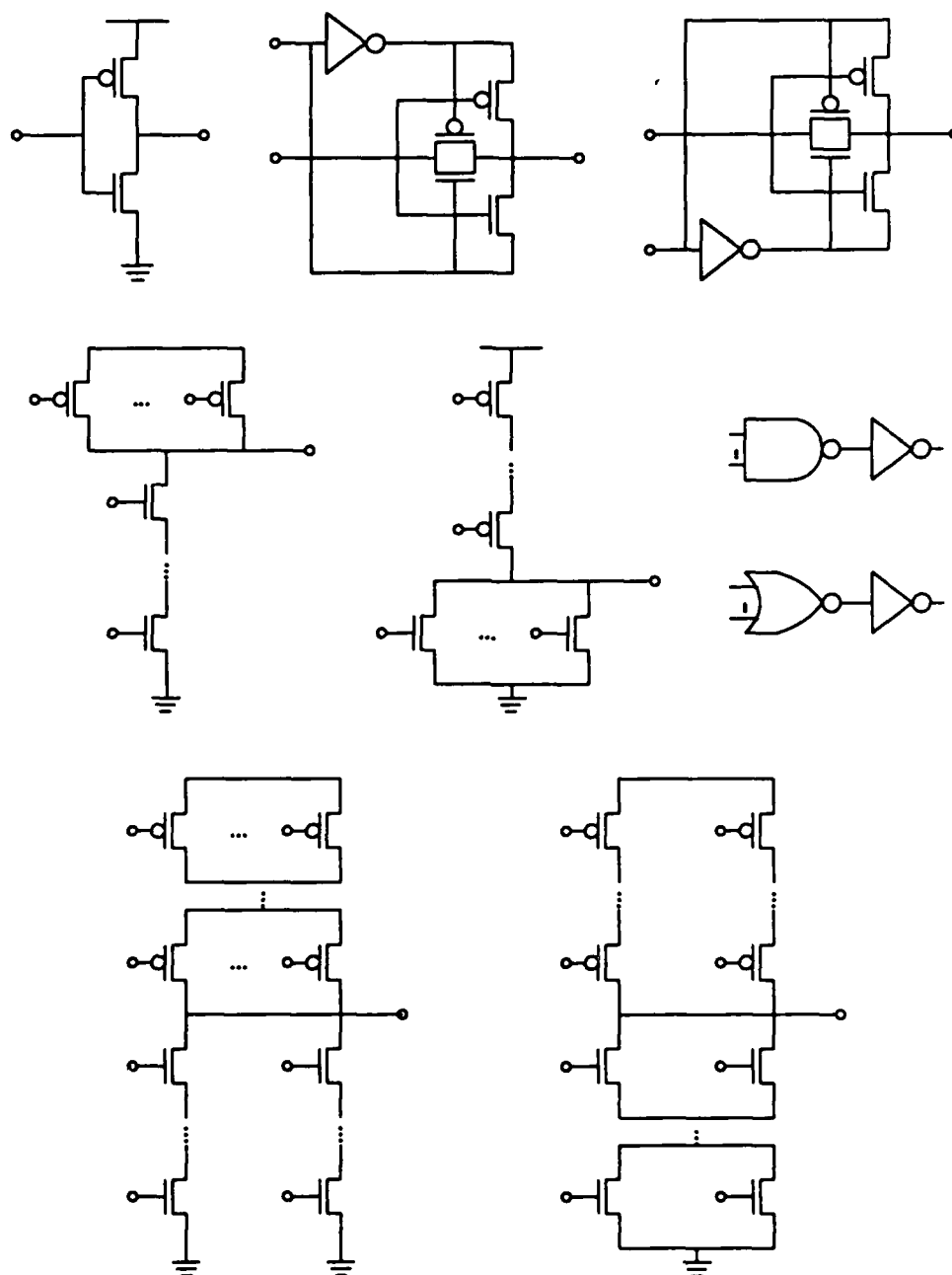


Figure 5.2. iSITE Predefined Library of Gates

than the circuits in a standard cell environment because the circuits are generated as needed and optimized automatically using iCOACH [Chen88a] (to be discussed later). Thus, these definitions will remain valid across technology updates. Hence, a large library of cells can be maintained without the disadvantages associated with a large standard cell library [Keut87a].

5.3. Circuit grouping and partitioning

The packing density of the layouts generated using the cell generator presented in Chapter 3 may decrease as the number of transistors becomes too large. This is partly due to the higher cost of intra-cell routing for larger cells. As the number of transistors and signals increases, many of the transistors and the gate signals must be placed farther apart, thus increasing the span of these nets. This makes it more difficult for several nets

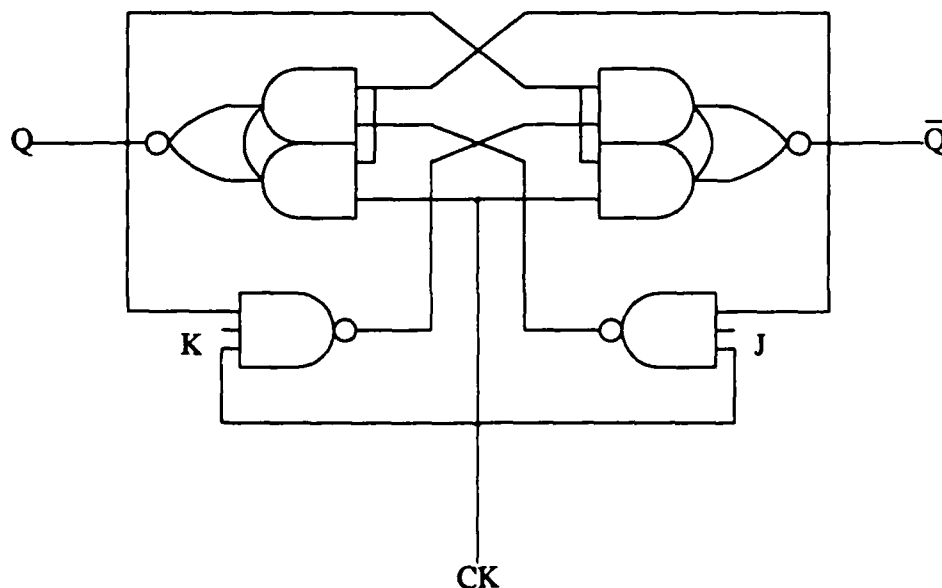


Figure 5.3. JK FLIP-FLOP

to share a track, because nets with overlapping spans must be assigned to different tracks. Two approaches can be used to minimize this problem. First, one can split the input signals, allowing the signal to have multiple entry points to the cell. This will reduce the span of individual nets, which will increase the number of nets sharing a track. Second, one can partition large circuits into several subcircuits which can be laid out efficiently by the cell generator. In this thesis, the latter approach is adopted. Partitioning reduces the problem size, which reduces the maximum memory requirements of the generator and reduces the total running time of the cell generator.

Before the circuit is partitioned, the transistors are grouped into indivisible blocks, to avoid the possibility of adding high parasitic capacitance on an internal gate node from intercell routing. Three different methodologies are used for grouping the transistors, namely, two user specified methods and one automatic method. The first two grouping methods depend on the user specified hierarchy. The first grouping method forms groups out of the transistors specified in the the lowest level of the hierarchy. These subcircuits usually correspond to logic gates. The second method places all transistors in the top-most level of the hierarchy together. These blocks are generally large enough to be generated as separate cells. Finally, the last method treats all channel connected transistors as indivisible blocks.

Channel connected transistor blocks correspond to logic gates, complex gates, and pass-transistor blocks. The channel connected transistor blocks are all the connected subgraphs in $G_{cc}(V,E)$. Graph G_{cc} is the same as the circuit graph, except that all voltage source nodes are split, so that each subcircuit is not connected through voltage source

nodes. An example of channel connected transistor blocks and the corresponding channel connected graph is shown in Figure 5.4.

The number of transistors in each channel-connected transistor block is generally small for random logic circuits. Since the cell generator is most efficient at generating

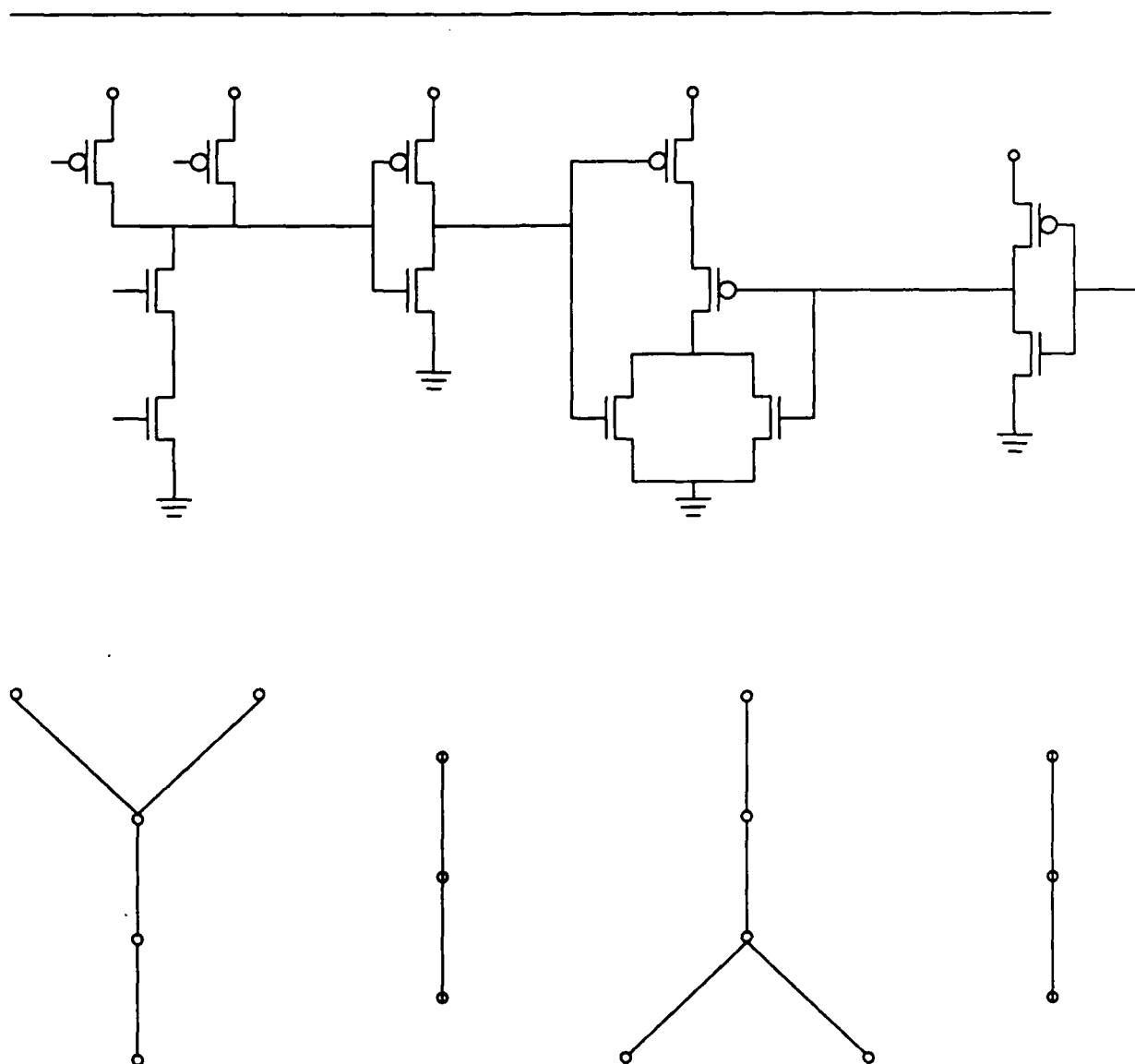


Figure 5.4. Channel-Connected Transistor Blocks

medium sized cells, several channel-connected transistor blocks are merged into medium-sized cells. Four algorithms have been developed to determine the transistor grouping.

The first algorithm uses a top-down approach to determine the transistor grouping. The set of channel-connected transistor blocks is recursively divided into two sets, such that the number of nets with terminals in both sets is minimized, until the sets are small enough. A min-net-cut heuristic has been chosen for two reasons. First, minimizing the number of nets between both sets minimizes the inter-cell routing area, and second, minimizing the inter-cell routing area tends to minimize the parasitic capacitance.

Algorithm 2 attempts to partition the set of transistor blocks into as few cells as possible, such that all cells contain less than a user specified quantity. Initially, all cells are empty. The set of transistor blocks is sorted by size in non-increasing order. Then, each block is assigned to the first cell which is large enough to hold the block. The running time of this algorithm is $O(n \log n + n^2) = O(n^2)$.

The third and fourth algorithms balance the two criteria of minimizing inter-cell connections on the one hand and generating as few cells as possible on the other. In both of these algorithms, the set of blocks that are candidates to be assigned to the current cell are blocks which contain nets with terminals in the current cell. In the third algorithm, the largest block that will fit in the current cell is selected. If there are no blocks connected to the current cell, then the largest free block that will fit is selected. This process of selecting blocks is repeated until the current cell is full. Then a new cell is started, and the process is repeated until all blocks have been assigned. The fourth algorithm is the

same as Algorithm 3 except that instead of choosing the largest block, the block with the most nets in common with the current cell is chosen.

5.4. Circuit extraction, recognition, and optimization

After the circuit has been geometrically laid out, the circuit parameters are extracted using iCPEX [Su87a]. The output from iCPEX is a flat SPICE description. This representation of the circuit is not suitable as input to iCOACH [Chen88a] to perform circuit optimization, since iCOACH is a gate-level optimization program.

The transistor specification must first be grouped into gates, such as NANDs, NORs, and AOI gates. The general problem of subcircuit recognition is equivalent to graph isomorphism and is NP-complete. However, this problem is not intractable for a wide range of CMOS circuits. For the following discussion, assume that the circuit does not contain any pass-transistor networks (this assumption will later be relaxed) and that the gates are composed of parallel or series connections.

Since this circuit is not composed of pass-transistor gates, a channel-connected transistor decomposition such as the method discussed in Section 5.2 will partition the set of transistors into a set of gates. Each subcircuit can be recognized independently and is composed of a small number of transistors, due to the electrical limitations of the devices (in general, there are fewer than six serially connected transistors).

The subcircuits at this stage consist only of series or parallel connections and so can be recognized efficiently using a super-transistor decomposition method [Acun89a]. In the super-transistor decomposition, a set of parallel or series gates is replaced with a single transistor. For example, the parallel PMOS transistors in Figure 5.5(a) are replaced

with the super PMOS transistor α in Figure 5.5(b). Then the series NMOS transistors are replaced with the super NMOS transistor α in Figure 5.5(c). This subcircuit can now be easily classified as a super inverter. The subcircuit function is

$$F = \bar{\alpha} = \overline{ABC}$$

or a three-input NAND gate. Of course for complex gates, this technique can be applied iteratively, replacing combinations of super-transistors with a single super-transistor. This method can recognize all NAND, NOR, INV, AOI, and OAI gates in time $O(n^3)$, where n is the number of transistors in the current subcircuit.

The original assumption that the circuit does not contain any pass-transistor networks is too restrictive. Commonly used pass-transistor gates are the XOR and XNOR

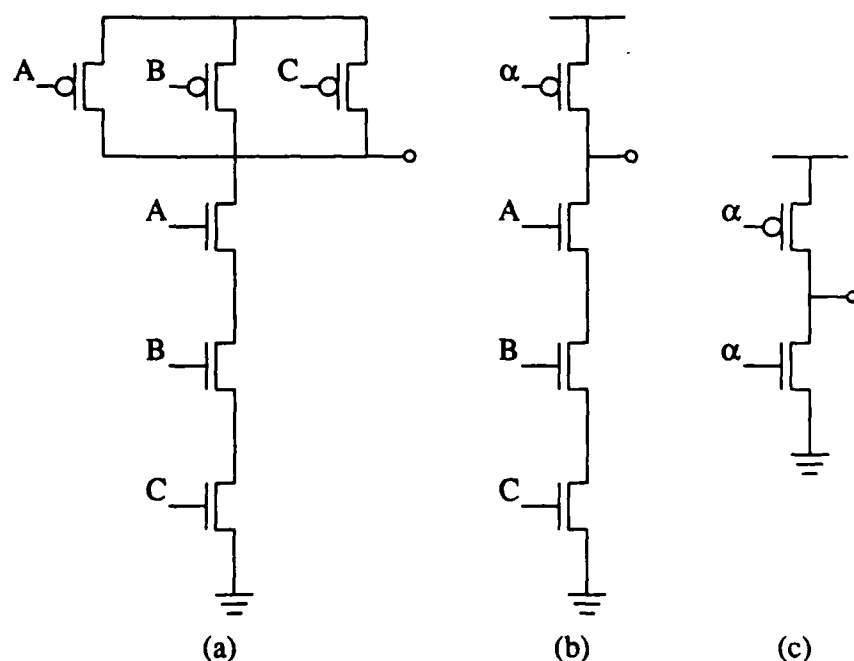


Figure 5.5. Super-Transistor Decomposition

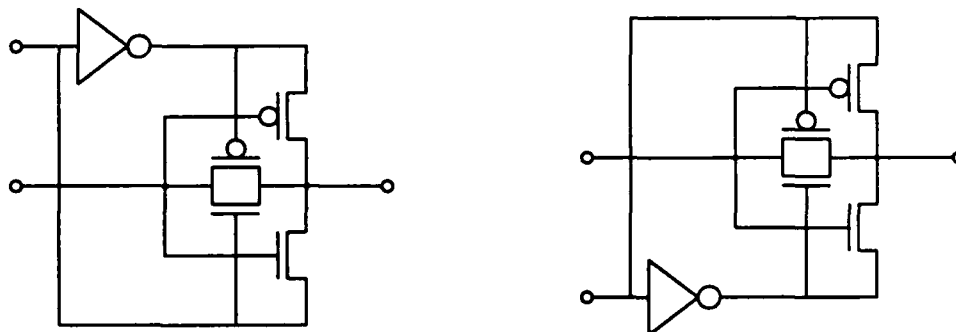


Figure 5.6. XOR and XNOR Pass Transistor Gates

gates shown in Figure 5.6. One problem with including pass-transistor gates in the circuit is that the simple channel-connected decomposition of Section 5.1 will not separate the pass-transistors from the static gates, as shown by the circuit graph in Figure 5.7. However, pass-transistors can be detected by using a technique that has been developed for fast-timing simulation [Over89a, Rao88a], as shown in Appendix A.

Another problem with the XOR and the XNOR gates is that they cannot be recognized using the super-transistor decomposition method. However, the XOR and the XNOR gates are composed of three easily recognizable transistor pairs, namely, inverter, transmission gate, and driver transistors. All transistor pairs can be recognized in time $O(n^2)$, where n is the number of transistors. Thus, the time to find all XOR and XNOR gates is $O(n^2 + \frac{n^2}{6}) = O(n^2)$.

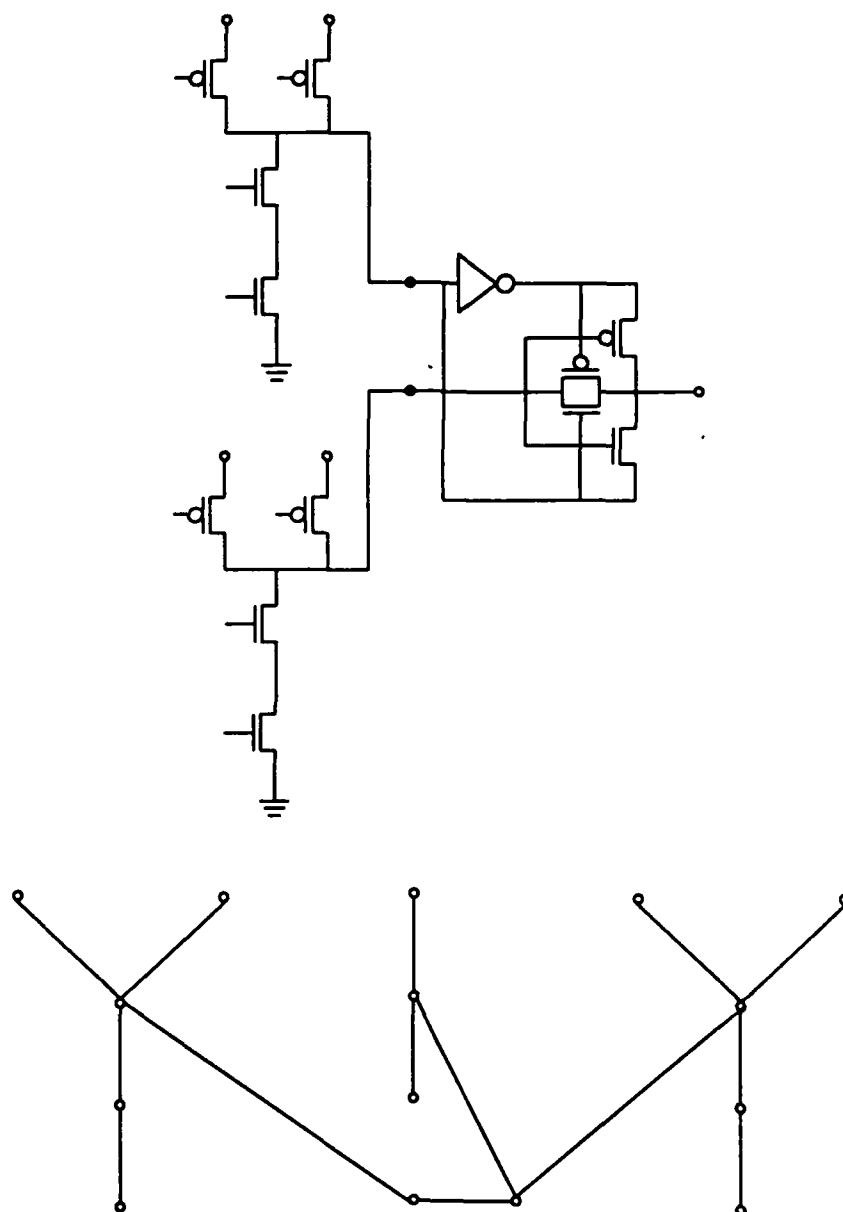


Figure 5.7. Channel Connected Transistors

5.5. Results

The iSITE system has been used to lay out a 32-bit adder with carry-look-ahead. The SPICE description of this circuit is shown in Appendix B and is implemented using 1254 transistors. Table 5.1 compares the four partitioning strategies presented in Section 5.3. In this table the active area for each layout is calculated as the maximum row width multiplied by the sum of the maximum cell height in each row. As expected, the first algorithm required the fewest tracks for intercell routing. The subcircuits varied greatly in size making it more difficult to generate rows of cells with the same height. However, this was offset by the area saved from inter-cell routing and produced the densest layout.

The second algorithm, which ignores connectivity information, generated blocks nearly equal in size. Although the layout generated by this algorithm required the smallest active area, it required the largest total cell area. This layout suffers on a global level, requiring the largest area for intercell wiring, using almost three times as many routing tracks as the previous algorithm.

The third and fourth algorithms are roughly equivalent. Algorithm 4, which placed more emphasis on connectivity than Algorithm 3 generated a slightly denser layout. In

Table 5.1. A Comparison of Partitioning Strategies

Algorithm	Routing Tracks	Active Area ($\times 10^6 \text{ } \mu\text{m}^2$)	Total Area ($\times 10^6 \text{ } \mu\text{m}^2$)
1	37	3.55	4.58
2	90	3.14	6.82
3	69	3.89	6.47
4	59	3.73	5.96

this example, the intercell routing area was the dominant factor to the cell area. The number of routing tracks varied by a factor of three, while the active area varied only by 18 percent.

CHAPTER 6.

CONCLUSIONS AND FUTURE RESEARCH

6.1. Conclusions

This thesis has presented a double-metal CMOS circuit synthesis tool. The iSITE system frees the designers from the tedious low-level aspects of design. The iSITE system manages the geometric details of layout design, so that the engineer can concentrate his or her efforts on circuit design or logic design. The iSITE system can easily generate layouts of complex transistor topologies, which previously could only be used in a full-custom design approach. Subcircuits are laid out dynamically as needed; thus designs can be generated in the latest technology available or in several different MOS technologies. The iSITE system has a built-in cell library composed of several classes of simple and complex logic gates. Additional subcircuits can be easily defined and maintained since only the transistor connectivity is needed. Unlike standard cell libraries, this library of subcircuits will not become obsolete with advances in technology.

The iSITE system also allows nested subcircuit definitions, where circuits can be defined in a hierarchal and structured fashion. This facilitates using subcircuits from other projects, since the electrical nodes visible to other subcircuits can be easily controlled. Thus, using these circuits requires knowledge only of its interfacing requirements.

The iSITE system first groups transistors together into indivisible units. These units are derived from a user specified hierarchy or from channel-connected decomposition. These indivisible units are then merged together to form larger cells for layout. The cells are represented by incidence matrices. The incidence matrix representation is not unique and the resulting layout is highly dependent on the incidence matrix ordering. In this thesis the incidence matrix is reordered to have the minimum total 1-1 distance. The problem of determining the matrix with the minimum total 1-1 distance is NP-complete, so a heuristic approach is employed. A symbolic layout of the circuit is then generated from the compacted incidence matrix. The physical mask data can then be generated from the symbolic layout along with the design rules for the fabrication process. The parasitic circuit parameters are then extracted from the layout using iCPEX. This transistor-level specification of the circuit is then converted into a gate-level representation for circuit optimization.

Subcircuit recognition is not only an important tool that is used to simplify circuit optimization, but can also be used to simplify circuit verification. For instance, this implementation of the 74181 4-bit ALU requires 258 transistors, which the subcircuit recognition program reduced to 31 simple and complex logic gates. For larger circuits, the logic gates should be grouped into larger functional blocks, such as flip-flops, registers, multiplexors, etc.

The new transistor sizes from circuit optimization along with the symbolic layout are used to generate new mask data. The cells are then placed using TimberWolf and interconnected using a channel router (YACR).

6.2. Future research

The incidence matrix compaction process operates on the columns first and then the rows, by recursively applying a min-net-cut algorithm to order both the rows and the columns. However, when the columns are merged together, additional constraints are imposed on the rows which may reduce the number of rows that can be merged together. Since the row and column ordering is determined by recursively partitioning the rows (columns), a complete ordering of the rows (columns) does not need to be determined before the columns (rows) are ordered. For instance, the min-net-cut algorithm could be applied alternately to the rows and columns. In general, reordering the rows tends to reduce the width of the cell and reordering the columns tends to reduce the height of the cell. Thus, better control of the final height or width of the cell can be obtained by judiciously choosing the order of the row and column optimization steps. A topic for future research would be to determine a better sequence of row and column optimization steps based on the aspect ratio, height, or width of the cell.

Three-and four-layer metal technology is currently being used in gate array designs. The symbolic layout generator should be extended to use additional layers automatically to reduce the cell area or to reduce parasitic resistances and capacitances. The symbolic layout generator should have the ability to stretch layouts to a specified height or width. For these constrained layouts, the multi-layer contacts and transistors should be oriented to achieve the specified height or width.

Currently all gate signals enter from the top or bottom of the cell. Thus, area is wasted in routing a local signal to the edge of the cell. The packing density of M^3

circuits could be further improved by folding the local signals. In addition, higher packing densities can be achieved by allowing inputs and outputs on all four sides of the cell. Circuits, such as bit-slice architectures and systolic arrays can easily take advantage of four-sided input/output by using one direction for control signals and the other direction for data. This generalization can easily be incorporated into the iSITE system without modification to the column ordering algorithm. These new nets can be represented in the column ordering algorithm, by connecting all the nets that enter/exit on the left side of the cell to the left pseudo node and connecting all nets that enter/exit on the right side of the cell to the right pseudo node. The column ordering algorithm is now applied to these new nets in conjunction with the nets derived from the incidence matrix.

Although this thesis has shown that laying out the transistors in alternate NMOS and PMOS transistor rows is undesirable, grouping the transistors into three or four regions can be advantageous. By using three or more regions, many signals can be folded together to reduce the layout area. An algorithm should be developed to determine the impact of adding an additional group to the layout.

The input to the iSITE system allows the user to specify the circuit hierarchically. However, iSITE flattens this hierarchy before generating the layout. The system should recognize common subcircuits from a user specified hierarchy or automatically from a flat circuit description and generate one layout for each common subcircuit that will be instantiated several times. This can significantly reduce the memory requirements and computation time necessary to generate large circuits.

Finally, the iSITE system can lay out arbitrary transistor topologies. However, the optimization program is limited to logic gates. This program should be extended to handle arbitrary transistor configurations since a significant reduction in circuit area can result by applying advanced transistor-level minimization techniques.

APPENDIX A.

PASS TRANSISTOR DETECTION

The following algorithm from iDSIM [Over89a] will detect pass transistors in channel connected transistor groups.

```

PASS_DETECT(  $\Omega$  ) {
    for (each SiEL $\Omega$ ) {
        /* detect CMOS pass transistors */
        for (each Tj, TkELSiSTTYPE(Tj) = n-type and TYPE(Tk) = p-type)
            if (Tj and Tk are connected in parallel) {
                MTYPE(Tj)GETSPASS;
                MTYPE(Tk)GETSPASS;
                if (NTYPE(SOURCE(Tj))  $\neq$  INPUT)
                    NTYPE(SOURCE(Tj))GETSPASS_OUT;
                if (NTYPE(DRAIN(Tj))  $\neq$  INPUT)
                    NTYPE(DRAIN(Tj))GETSPASS_OUT;
            }

        /* flag NMOS load output nodes */
        for (each TjELSiSTMTYPE(Tj) = LOAD)
            NTYPE(GATE(Tj))GETSDRIVEN;

        /* flag CMOS gate output nodes */
    }
}

```

for (each N_j EL Si ST N_j has a n-type DRIVER transistor and
a p-type DRIVER transistor adjacent)

NTYPE(N_j)GETSDRIVEN;

/* check for obvious pass transistors */

for (each T_j EL Si ST MTYPE(T_j) = DRIVER)

if (TYPE(T_j) = n-type) {

if (SOURCE(T_j) = V_{DD}) {

NTYPE(DRAIN(T_j))GETSPASS_OUT;

MTYPE(T_j)GETSPASS;

}

else if (DRAIN(T_j) = V_{DD}) {

NTYPE(SOURCE(T_j))GETSPASS_OUT;

MTYPE(T_j)GETSPASS;

}

}

else { /* p-type */

if (SOURCE(T_j) = GND) {

NTYPE(DRAIN(T_j))GETSPASS_OUT;

MTYPE(T_j)GETSPASS;

}

else if (DRAIN(T_j) = GND) {

NTYPE(SOURCE(T_j))GETSPASS_OUT;

```

        MTYPE(Tj)GETSPASS;
    }
}

/* check for path between DRIVEN nodes */
if (Si has more than one DRIVEN node) {
    for (each DRIVER transistor Tj in a path between
        DRIVEN nodes)
        MTYPE(Tj)GETSPASS;
    for (each INTERNAL node Nj in a path between DRIVEN nodes)
        NTYPE(Nj)GETSPASS_OUT;
}

if (Si has at least one PASS_OUT node) {
    /* check for paths to GND and VDD from PASS_OUT nodes */
    PASS_CHECK(Si, GND, n-type);
    PASS_CHECK(Si, VDD, p-type);
}

for (each DRIVER transistor Tj in a path between DRIVEN and
    PASS_OUT nodes)
    MTYPE(Tj)GETSPASS;
for (each INTERNAL node Nj in a path between DRIVEN and
    PASS_OUT nodes)
    NTYPE(Nj)GETSPASS_OUT;
}

```

```

}
```

```

PASS_CHECK( Si, source, type ) {
```

```

    /* check for paths from the source to PASS_OUT nodes */
```

```

    for (each path between source and each PASS_OUT node in Si)
```

```

        if (each node in the path is INTERNAL and each transistor is a
```

```

            type DRIVER transistor) {
```

```

                for (each transistor Tj in the path)
```

```

                    MTYPE(Tj)GETSPASS;
```

```

                for (each node Nj in the path)
```

```

                    NTYPE(Nj)GETSPASS_OUT;
```

```

            }
```

```

    /* check for paths between PASS_OUT nodes */
```

```

    for (each path between PASS_OUT nodes in Si)
```

```

        if (each node in the path is INTERNAL and each transistor is a
```

```

            type DRIVER transistor) {
```

```

                for (each transistor Tj in the path)
```

```

                    MTYPE(Tj)GETSPASS;
```

```

                for (each node Nj in the path)
```

```

                    NTYPE(Nj)GETSPASS_OUT;
```

```

            }
```

```

}
```

APPENDIX B.

32-BIT CARRY-LOOK-AHEAD ADDER

The following SPICE description implements the 32-bit carry-look-ahead adder using 1254 transistors (Chapter 5).

```
xb00 A0 B0 CIN P0 G0 S0 bitadd32
xb01 A1 B1 C0 P1 G1 S1 bitadd32
xb02 A2 B2 C1 P2 G2 S2 bitadd32
xb03 A3 B3 C2 P3 G3 S3 bitadd32
xb04 A4 B4 C3 P4 G4 S4 bitadd32
xb05 A5 B5 C4 P5 G5 S5 bitadd32
xb06 A6 B6 C5 P6 G6 S6 bitadd32
xb07 A7 B7 C6 P7 G7 S7 bitadd32
xb08 A8 B8 C7 P8 G8 S8 bitadd32
xb09 A9 B9 C8 P9 G9 S9 bitadd32
xb10 A10 B10 C9 P10 G10 S10 bitadd32
xb11 A11 B11 C10 P11 G11 S11 bitadd32
xb12 A12 B12 C11 P12 G12 S12 bitadd32
xb13 A13 B13 C12 P13 G13 S13 bitadd32
xb14 A14 B14 C13 P14 G14 S14 bitadd32
xb15 A15 B15 C14 P15 G15 S15 bitadd32
```

xb16 A16 B16 C15 P16 G16 S16 bitadd32

xb17 A17 B17 C16 P17 G17 S17 bitadd32

xb18 A18 B18 C17 P18 G18 S18 bitadd32

xb19 A19 B19 C18 P19 G19 S19 bitadd32

xb20 A20 B20 C19 P20 G20 S20 bitadd32

xb21 A21 B21 C20 P21 G21 S21 bitadd32

xb22 A22 B22 C21 P22 G22 S22 bitadd32

xb23 A23 B23 C22 P23 G23 S23 bitadd32

xb24 A24 B24 C23 P24 G24 S24 bitadd32

xb25 A25 B25 C24 P25 G25 S25 bitadd32

xb26 A26 B26 C25 P26 G26 S26 bitadd32

xb27 A27 B27 C26 P27 G27 S27 bitadd32

xb28 A28 B28 C27 P28 G28 S28 bitadd32

xb29 A29 B29 C28 P29 G29 S29 bitadd32

xb30 A30 B30 C29 P30 G30 S30 bitadd32

xb31 A31 B31 C30 P31 G31 S31 bitadd32

x01 CIN G0 P0 G1 P1 G2 P2 G3 P3 C0 C1 C2 G0P1 P0P1 bcla

x02 C3 G4 P4 G5 P5 G6 P6 G7 P7 C4 C5 C6 G1P1 P1P1 bcla

x03 C7 G8 P8 G9 P9 G10 P10 G11 P11 C8 C9 C10 G2P1 P2P1 bcla

x04 C11 G12 P12 G13 P13 G14 P14 G15 P15 C12 C13 C14 G3P1 P3P1 bcla

x05 C15 G16 P16 G17 P17 G18 P18 G19 P19 C16 C17 C18 G4P1 P4P1 bcla

x06 C19 G20 P20 G21 P21 G22 P22 G23 P23 C20 C21 C22 G5P1 P5P1 bcla

x07 C23 G24 P24 G25 P25 G26 P26 G27 P27 C24 C25 C26 G6P1 P6P1 bcla

x08 C27 G28 P28 G29 P29 G30 P30 G31 P31 C28 C29 C30 G7P1 P7P1 bcla

x09 CIN G01 P01 G11 P11 G21 P21 G31 P31 C3 C7 C11 G0P2 P0P2 bcla

x10 C15 G41 P41 G51 P51 G61 P61 G71 P71 C19 C23 C27 G1P2 P1P2 bcla

x11 P0P2 CIN G0P2 C15 aoi21

x12 CIN P0P2 P1P2 P1P2 G0P2 G1P2 C31 aoi321

* combine bit for 32-bit alu -> bitadd32(A,B,C,P,G,S)

x1 A B P xor

x2 A B G and2

x3 C P S xor

* carry look ahead

x1 cin p0 g0 c0 aoi21

x2 cin p0 p1 g0 p1 g1 c1 aoi321

x3 cin p0 p1 p2 g0 p1 p2 g1 p2 g2 c2 aoi4321

x4 p3 p2 p1 g0 p3 p2 g1 p3 g2 g0p1 aoi432

x5 p3 p2 p1 p0 p0p1 and4

APPENDIX C.

AN NP-COMPLETE PROBLEM -- MINIMUM TOTAL 1-1 DISTANCE

In this Appendix, it will be shown that finding a permutation of the rows of an incidence matrix with a minimum total 1-1 distance is NP-complete. To prove the NP-completeness of this problem it is sufficient to show that the following subproblem is NP-complete.

Minimum Total 1-1 Distance Decision

Given an incidence matrix A_n and a positive integer K_m , determine

if there exists a row permutation matrix P such that

$$\text{total 1-1 distance } (PA_n) \leq K_m.$$

To prove that the minimum total 1-1 distance decision problem is NP-complete, it will be shown that the minimum total 1-1 distance is in the class NP. Then a polynomial transformation of the well-known NP-complete problem, namely, the optimal linear arrangement problem [Gare79a], to the minimum total 1-1 distance problem will be constructed.

The optimal linear arrangement problem can be stated as:

Optimal Linear Arrangement Decision

Given a graph $G(V,E)$ and a positive integer K_o , determine

if there exists a bijection $f:V \rightarrow \{1, \dots, n\}$, where $|V| = n$, such that

$$\sum_{(v_i, v_j) \in E} |f(v_i) - f(v_j)| \leq K_0.$$

The minimum total 1-1 distance problem is in class NP, since a nondeterministic algorithm can guess a permutation matrix P_0 and compute the total 1-1 distance of $P_0 A_a$ in polynomial time using the method described in Section 2.

To transform graph G into an incidence matrix, let each edge $d_k = (v_i, v_j) \in E$ define a column vector $\hat{c}_k = \hat{e}_i + \hat{e}_j$, where \hat{e}_x is a standard basis vector of field F^n . So, $e^t_1 = (1, 0, \dots, 0)$, $e^t_2 = (0, 1, \dots, 0)$, $e^t_n = (0, 0, \dots, 1)$. Construct an $n \times m$ matrix ($|E| = m$) $A = [\hat{c}_1 \hat{c}_2 \cdots \hat{c}_m]$. The matrix A is an incidence matrix since each column contains exactly two 1s.

Now the validity of the above transformation will be proved by showing the equivalence between the following two statements:

- (1) there is a bijection $f: V \rightarrow \{1, \dots, n\}$ such that $\sum_{(v_i, v_j) \in E} |f(v_i) - f(v_j)| \leq K_0$, and
- (2) there is a row permutation matrix P such that the total 1-1 distance $(PA_a) \leq K_m$.

Proof 1 \rightarrow 2:

Let the row permutation matrix $P = \begin{bmatrix} e^t_{g(1)} \\ e^t_{g(2)} \\ \vdots \\ e^t_{g(n)} \end{bmatrix}$, where $g(x) = f(v_x)$. Then the matrix PA_a is

given by

$$PA_a = \begin{bmatrix} e^t_{g(1)} \hat{c}_1 & e^t_{g(1)} \hat{c}_2 & \cdots & e^t_{g(1)} \hat{c}_m \\ \vdots & \vdots & \ddots & \vdots \\ e^t_{g(n)} \hat{c}_1 & \cdots & \cdots & e^t_{g(n)} \hat{c}_m \end{bmatrix}$$

Consider column \hat{c}_k with 1s in the i^{th} and j^{th} entries. Notice that $e^t_{g(x)} \hat{c}_k$ is the x^{th} entry of

\hat{c}_k , so the 1-1 distance of column \hat{c}_k is

$$|g(i) - g(j)| = |f(v_i) - f(v_j)|.$$

But the cost of the corresponding edge d_k is $|f(v_i) - f(v_j)|$. Therefore the

$$\text{total 1-1 distance } (PA_a) = \sum_{(v_i, v_j) \in E} |f(v_i) - f(v_j)| \leq K_0. \square$$

Proof of 2 \rightarrow 1

Let the $n \times n$ row permutation matrix $P = \begin{bmatrix} \hat{p}_1 \\ \hat{p}_2 \\ \vdots \\ \hat{p}_n \end{bmatrix}$. Then for each \hat{p}_x there exists one \hat{e}_y

such that $(\hat{p}_x \cdot \hat{e}_y) = 1$, since P is a row permutation matrix. Define the function $g(x) = \{ y \mid (\hat{p}_x \cdot \hat{e}_y) = 1 \}$. In other words, g is a mapping of the rows of A_a to the rows of PA_a . So for each column \hat{c}_k with 1s in the i^{th} and j^{th} entries, the permuted column $P\hat{c}_k$ has 1s in row $g(i)$ and row $g(j)$. The 1-1 distance of $P\hat{c}_k$ is $|g(i) - g(j)|$. Now define the function $f(v_x) = g(x)$. Then the cost of edge $d_k = (v_i, v_j)$ is

$$|f(v_i) - f(v_j)| = |g(i) - g(j)| = \text{1-1 distance}(P\hat{c}_k).$$

But, this is true for all k , so

$$\sum_{(v_i, v_j) \in E} |f(v_i) - f(v_j)| = \text{total 1-1 distance } (PA_a) \leq K_m. \square$$

APPENDIX D.

iSITE USER'S MANUAL

D.1. Introduction

The iSITE system automatically generates MOS circuits in the metal-metal matrix (M^3) layout style. iSITE is written in C and runs under the UNIX operating system. The iSITE system consists of a suite of programs, each performing a specific layout task. The circuits are specified in SPICE format, with several pre-defined built-in logic gates. This transistor description is partitioned into channel-connected components. Several of these components are then merged together to form larger cells. Each of these cells is then converted into a symbolic layout. The symbolic layout can then be converted into either M^3 variation. After the mask layers for all the cells have been generated, the cells are placed and routed using TimberWolfSC [Sech86a] and YACR [Reed85a]. The circuit parameters can then be extracted using iCPEX [Su87a]. This transistor description is then converted into a gate description for circuit optimization using iCOACH [Chen88a]. The new transistor sizes along with the symbolic layout can then be used to generate a new physical layout.

This appendix is organized as follows. Section D.2 begins with an overview of the iSITE circuit synthesis process. Sections D.2.1 through D.2.6 describe each of the iSITE programs in detail. In this description, "[]" and "[]" represent optional arguments, while " | " between arguments represents a choice of several arguments. Section D.2.1

describes using the iSPICE2AA program to convert circuits represented in SPICE format into compacted incidence matrices. A symbolic layout can then be generated directly from these incidence matrices with iLAYOUT, or the height of the matrix can be adjusted with the iPREFOLD program in Section D.2.2. iPREFOLD analyzes the incidence matrix and removes vertical constraints by adding additional columns to the layout. The next section presents iLAYOUT, a program to generate a symbolic layout from compacted incidence matrices. First the command line arguments are described, followed by a description of the input format. In Section D.2.4, the symbolic layout is converted into the physical mask layers by iSILVER. Section D.2.5 presents iD2GATE for converting a transistor-level description into a gate-level description. In the iSITE system, iD2GATE is used to convert the output of iCPEX, into a form suitable for circuit optimization by iCOACH. Finally, Section D.2.6 concludes with iUPDTRAN, a program which updates the transistor sizes in the symbolic layout with the values obtained from iCOACH.

D.2. Circuit synthesis with iSITE

The following steps provide a framework for generating M^3 layouts with the iSITE system:

1. represent circuit in SPICE format,
2. use iSPICE2AA to convert SPICE into compacted incidence matrices,

3. use iPREFOLD to perform prefolding analysis (optional),
4. use iLAYOUT to convert compacted incidence matrices into symbolic layouts,
5. generate physical layout with iSILVER,
6. use iCPEX to extract circuit parameters,
7. use iD2GATE to convert transistor-level description into a gate-level description,
8. use iCOACH to optimize transistor sizes,
9. update the transistor sizes and generate new layouts,
10. place cells using TimberWolfSC,
11. interconnect cells using YACR.

D.2.1. Generating compacted incidence matrices (iSPICE2AA)

The program iSPICE2AA converts a SPICE description of a circuit into a set of compacted incidence matrices. iSPICE2AA accepts a complete SPICE description of a circuit, including nested subcircuit definitions. However, the circuit is flattened internally. iSPICE2AA has several built-in logic gates, namely, INV, XOR, XNOR, NAND n , NOR n , AND n , OR n , AOI $n_1 n_2 \cdots n_k$, and OAI $n_1 n_2 \cdots n_k$, where n and n_k represent the number of inputs. The format of the built-in logic gates is

Xname inputs output "logic gate."

For example, the logic function,

$$F = \overline{abc+de+f+ghi},$$

can be implemented with the following AOI gate,

X1 a b c d e f g h i F AOI3213.

iSPICE2AA uses only the transistor connectivity to generate incidence matrices. In other words, the width and length information on transistor cards is ignored. Also, all resistors are treated as ideal wires and all capacitors are ignored. In addition, all nonzero dc voltage sources are treated as V_{dd} . iSPICE2AA accepts the following options.

SYNOPSIS

iSPICE2AA [-12AHOefmv] [-g[n]] [-M[n|p]N] [-o outfile] [-p prefix] [-s size] file

The options to the iSPICE2AA program are interpreted in the following manner.

Column and Row Assignment Options

-1

This option forces one column to be filled before starting the next column.

-2

This option sorts all transistors according to their drain nodes and assigns each transistor to the first available column.

-O

This option will allow only series transistors to be allocated to the same column.

-MnN

This option will allow at most N NMOS transistors to be merged together in a single column.

-MpN

This option will allow at most N PMOS transistors to be merged together in a single column.

-v

This option will force V_{dd} and V_{ss} to be the first and last rows in the incidence matrix, respectively.

Grouping Options**-g0**

This option will group all transistors into one indivisible group. The -s option is ignored.

-g1

This option will divide the circuit into channel-connected groups.

-g2

The indivisible transistor groups are defined by the first-level subcircuit definition.

-g3

The transistors are grouped according to the bottommost level of subcircuits. This grouping strategy generally groups the transistors by the user specified logic gates.

Packing Options**-P0**

This option specifies that the groups should not be merged together.

-P1

The criterion for merging groups is based solely on size. Since transistor connectivity is ignored, the merged groups are nearly equal in size. This method produces the fewest number of groups.

-P2

This packing method merges groups together by size and the number of nets in common with other groups. The group with the most nets is chosen as the seed. Then, only groups connected to this set are chosen. When the current merged group is large enough, a new block is started.

-P3

This packing method is similar to -P2, except that the largest block is chosen as the seed.

-P4

This option is reserved for future expansion.

-P5

This packing method ignores the size of the groups. The set of groups are partitioned recursively until the set is small enough, using a min-net-cut heuristic. This method generally produces cells which require the least area for inter-cell routing.

-s N

The indivisible groups are merged together to form groups with at most N transistors. Groups containing more than N transistors are not split.

Miscellaneous Options**-H**

This options prints an expanded help list.

file

The specified *file* describes the transistor connectivity in SPICE format.

D.2.2. Prefolding analysis (iPREFOLD)

The height of the symbolic layout can be controlled by merging fewer columns together, since merging columns impose additional constraints. The height of the symbolic layout is bounded from below by two quantities, namely, the maximum net-density and the length of the longest path in the vertical constraint graph. iPREFOLD takes a compacted incidence matrix as input and produces a list of constraints to break by adding additional columns to the layout. iPREFOLD accepts a single optional argument.

SYNOPSIS

iPREFOLD [file]

Options

file

The specified file is used as input.

D.2.3. Symbolic layout (iLAYOUT)

iLAYOUT produces a symbolic layout of a compacted incidence matrix, which can be obtained from a SPICE description by using the iSPICE2AA program. The columns in the symbolic layout (each column in the matrix and each implicit signal represent a column in the symbolic layout) are ordered by a modified min-net-cut algorithm which minimizes the net-density across columns, as opposed to between columns. The algorithm also considers the interaction between the PMOS and NMOS regions in CMOS circuits. After the columns are ordered, the nets are assigned to tracks using a left-edge first algorithm.

The format of the input file, with keywords in bold, is as follows:

[group integer]

[size integer]

[global

[string] ... [string]

]

[longabbr]

[order

[[PMOS column #][, NMOS column #] | [signal]] ...

]

[pplane | nplane

header

[signal] ...

Amatrix

[incidence matrix]

]

[label

[signal fullname]

]

[end] □

Lines beginning with a “#” are ignored. The signal names must be one character long

unless the command **longabbr** is used, in which case all signal names must be delimited by a comma.

The iLAYOUT program accepts the following command line options.

SYNOPSIS

iLAYOUT [-d[m][level]] [-beglmOsSTv] [-hn] [-tn] [file]

Options

-b

This option will try several heuristics and choose the best layout.

-dlevel

Set the general debugging *level*.

-dmlevel

Set the debugging *level* for the min-net-cut routines.

-e

This option will print the estimated height of the symbolic layout in rows.

-g group

Generate the symbolic layout for the specified group. The default is to generate a layout for the first group only.

-H

Print an expanded help list.

-h0

Use min-net-cut bisection between columns.

-h1

Use min-net-cut between columns.

-h2

Use min-net-cut bisection across all columns.

-h3

Use min-net-cut across all columns.

-h4

Use min-net-cut bisection across the k columns with the fewest nets.

-h5

Use min-net-cut across the k columns with the fewest nets.

-h6

Use min-net-cut bisection across the k columns with the most nets.

-h7

Use min-net-cut across the k columns with the most nets.

-l

Assign nets to tracks using a left-edge first algorithm.

-m

Do not merge nets in the layout.

-Mn

Set the maximum number of candidate to n for min-net-cut heuristics 4 and 5.

-O

Perform local column optimization.

-si

Split internal nets.

-sc

Split constraint nets.

-2

Split multi-terminal nets into two terminal nets.

-Sn

Set the min-net-cut set size to n .

-T

Generate table of transistor sizes and locations.

D.2.4. Physical layout (iSILVER)

The symbolic layout generated by iLAYOUT can be converted into a metal-metal matrix layout (M^3). Both variations of the M^3 style can be generated from the same

symbolic representation. The input format for iSILVER is as follows.

[global

 [signal] ...

]

[longabbr]

nrows rows ncols columns **prows rows pcols columns**

[layout

symbolic layout

]

[size

 [(x-coor y-coor width length)

]]

[label

 [signal fullname]

]

[xlabel

 type name x-coor y-coor layer

]

[end] □

SYNOPSIS

iSILVER [-d[level]] [-12abcEcklsTtv] [-r design_rules] [-o output] [file]

iSILVER accepts the following command line options.

Design Rule Options

-a

Use the ATT (3 micron, pwell) design rules.

-v

Use the VTI (2 micron, dual well) design rules.

-r *file*

Use the design rules in the specified *file*.

Extraction Options

-c

Add contacts to signal lines for extraction.

-E

Use iCPEX CIF point extension commands to label nodes and signals for extraction.

-e

Set all options applicable for extraction by iCPEX.

-t

Label the transistors.

Layout Options**-1**

Generate the physical layout in style 1 (lower metal layer is horizontal).

-2

Generate the physical layout in style 2 (upper metal layer is horizontal).

-b

Label signal columns at the bottom of the signal (smallest y-coordinate).

-k

Represent the physical layout in KIC format (default is CIF).

-l

Do not include the cell library in the physical layout.

TimberWolf Options**-s**

Save cell statistics.

-T

Generate a TimberWolf .cel file.

-Ta

Append to the TimberWolf .cel file when writing.

-T#val

Use *val* as the cell number in the TimberWolf .cel file.

-Tg

Ignore all local signals. The TimberWolf .cel file will only contain entries for the global signals.

-Tofile

Use the specified *file* as the TimberWolf .cel file.

Miscellaneous Options**-dlevel**

Set the general debugging *level*.

-H

Print an expanded help list.

-o file

Use the specified *file* for the physical layout.

D.2.5. Gate recognition (iD2GATE)

The transistor-level description produced by iCPEX and other circuit extractors is not suitable for input to iCOACH. iD2GATE will take as input a transistor-level description in SPICE format and output a gate-level description in SPICE or in a format compatible with iCOACH. The transistor-level description is grouped into channel-connected transistors and pass-transistor blocks using iDSIM [Over89a]. Each of these blocks is

then decomposed into its logic function using super-transistor decomposition and transistor pairing. The following options are recognized by iD2GATE.

SYNOPSIS

iD2GATE [-cv] [-i program] [-S file] [-s file] [-o file] file

Options

-c

Produce a gate-level description in iCOACH format. The default is to produce SPICE.

-i program

Execute *program* instead of iDSIM to detect pass transistors.

-o file

Use the specified file for output.

-S file

Use the specified file for statistics.

-s file

The pass transistors have already been detected and are in the specified file.

-v

Add voltage sources to all undriven transistor gate nodes. Note: iDSIM requires that a path exists between the gate of a transistor to a voltage source, or to a

transistor drain, or to a transistor source. Also, all voltage sources must be defined before any devices.

file

The specified file contains the transistor-level description in SPICE.

D.2.6. Updating transistor sizes (iUPDTRAN)

The task of updating the transistor sizes in the symbolic layout after executing iCOACH is both time-consuming and error prone. The program, iUPDTRAN, will automatically update the transistor sizes. The following options are recognized by iUPDTRAN.

SYNOPSIS

iUPDTRAN icpex.log icoach.sta sym.stat gate.stat

Arguments

icpex.log

This file specifies the physical location of all transistors. This file is produced by iCPEX.

icoach.sta

This file specifies the output from iCOACH.

sym.stat

This file maps the physical location of all transistors to their symbolic location and is produced automatically by iSILVER.

gate.stat

This file specifies which transistors have been replaced by gates and is automatically produced by iD2GATE.

REFERENCES

- [Acun89a] E. Acuna, "Hierarchical device recognition, characterization, and implementation for iSPLICE3," M.S. thesis, University of Illinois, Urbana-Champaign, 1989.
- [Berg70a] C. Berge, *Graphs and Hypergraphs*, North-Holland, London, 1970.
- [Bray84a] R. K. Brayton, N. L. Brenner, C. L. Chen, G. DeMichelli, C. T. McMullen, and R. H. J. M. Otten, "The yorktown silicon compiler," IBM Technical Report, 1984.
- [Bray84b] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. Sangiovanni-Vincentelli, "ESPRESSO-II: A new logic minimizer for programmable logic arrays," *Custom Integrated Circuit Conf.*, pp. 370-376, May 1984.
- [Bray84c] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, Hingham, MA, 1984.
- [Bray87a] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: A multiple-level logic optimization system," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 1062-1081, November 1987.
- [Chen88a] H. Y. Chen and S. M. Kang, "iCOACH: A circuit optimization aid for CMOS high-performance circuits," *IEEE Int. Conf. on Computer-Aided Design*, pp. 372-375, November 1988.
- [Chen88b] C. Y. R. Chen and C. Y. Hou, "A new VLSI layout optimization methodology for CMOS functional arrays," *31st Midwest Symp. on Circuits and Syst.*, pp. 996-999, August 1988.
- [Cowa87a] S. Coward, "The impact of two level metallization on PLA performance and the Genesis PLA generator," M.S. thesis, University of Illinois, Urbana-Champaign, 1987.
- [DeGe86a] A. J. DeGeus and W. Cohen, "A rule-based system for optimizing combinational logic," *IEEE Design and Test of Computers*, vol. 2, pp. 22-32, August 1986.

- [DeMi85a] G. DeMicheli, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Optimal state assignment for finite state machines," *IEEE Trans. Computer-Aided Design*, vol. CAD-4, no. 3, pp. 269-284, July 1985.
- [Deva86a] S. Devadas and A. R. Newton, "GENIE: A generalized array optimizer for VLSI synthesis," *23rd Design Automation Conf.*, pp. 631-637, 1986.
- [Doen87a] J. Doenhardt and T. Lengauer, "Algorithmic aspects of one-dimensional layout compaction," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, no. 5, pp. 863-878, September 1987.
- [Duml83a] D. Dumlugol, H. De Man, P. Stevens, and G. Schrooten, "Local relaxation algorithms for event-driven simulation of MOS networks including assignable delay modeling," *IEEE Trans. Computer-Aided Design*, vol. CAD-2, no. 3, July 1983.
- [Dunl80a] A. E. Dunlop, "SLIM -- The translation of symbolic layouts into mask data," *17th Design Automation Conf*, pp. 595-602, 1980.
- [Egan82a] J. Egan and C. L. Liu, "Optimal bipartite folding of PLAs," *19th Design Automation Conf.*, 1982.
- [Feld83a] S. I. Feldman, "The circuit design language Xi," *Int. Conf. on Computer Design*, pp. 652-655, 1983.
- [Gajs84a] D. D. Gajski and J. J. Bozek, "ARSENIC: Methodology and Implementation," *Int. Conf. on Computer-Aided Design*, pp. 116-118, November 1984.
- [Gare79a] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, Murray Hill, NJ, 1979.
- [Gee87a] P. Gee, M. Y. Wu, S. M. Kang, and I. N. Hajj, "Metal-metal matrix (M^3) CMOS cell generator with compaction," *IEEE Int. Conf. on Computer-Aided Design*, pp. 184-187, November 1987.
- [Gee88a] P. Gee, I. N. Hajj, and S.M. Kang, "iSILVER: A symbolic layout generator for MOS circuits," *Proc. of 31st Midwest Symp. on Circuits and Syst.*, pp. 142-145, August 1988.
- [Gee89a] P. Gee, M. Y. Wu, I. N. Hajj, S. M. Kang, and W. Shu, "Automatic circuit synthesis using switching network logic and metal-metal-matrix layout," in *Advances in Computer-Aided Engineering Design*: I. N. Hajj, ed., JAI Press, 1989.

- [Gee89b] P. Gee, I. N. Hajj, and S. M. Kang, "An improved min-net-cut approach for gate matrix and metal-metal matrix circuit layout," *Proc. of 32nd Midwest Symp. on Circuits and Syst.*, August 1989.
- [Gee89c] P. Gee, M. Y. Wu, I. N. Hajj, and S. M. Kang, "Automatic synthesis of metal-metal-matrix layout (M^3)," *to appear: Int. J. Computer-Aided VLSI Design*, 1989.
- [Gee89d] P. Gee, I. N. Hajj, and S. M. Kang, "A custom cell generation system for double metal CMOS technology," *IEEE Int. Conf. on Computer-Aided Design*, November 1989.
- [Gibs78a] D. Gibson and S. Nance, "SLIC -- symbolic layout of integrated circuits," *13th Design Automation Conf.*, pp. 289-295, May 1978.
- [Hach82a] G. D. Hachtel, A. R. Newton, and A. L. Sangiovanni-Vincentelli, "Techniques for programmable logic array folding," *19th Design Automation Conf.*, pp. 147-153, 1982.
- [Hajj83a] I. N. Hajj and D. Saab, "Fault modeling and logic simulation of MOS VLSI circuits based on logic expression extraction," *Int. Conf. on Computer-Aided Design*, pp. 99-100, September 1983.
- [Hash71a] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures," *Proc. 18th Design Automation Workshop*, pp. 155-168, 1971.
- [Hein88a] D. V. Heinbuch, *CMOS3 Cell Library*, Addison-Wesley Publishing Company, Inc., 1988.
- [Hong74a] S. J. Hong, R. G. Cain, and D. L. Ostapko, "MINI: A heuristic approach for logic minimization," *IBM J. Res. and Develop.*, pp. 443-458, September 1974.
- [Huer88a] J. L. Huertas and J. M. Quintana, "A new method for the efficient state-assignment of PLA-based sequential machines," *IEEE Int. Conf. on Computer-Aided Design*, pp. 156-159, November 1988.
- [Hwan87a] D. K. Hwang, W. K. Fuchs, and S. M. Kang, "An efficient approach to gate matrix layout," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, no. 5, pp. 802-809, September 1987.
- [Hwan86a] S. Y. Hwang, R. W. Dutton, and T. Blank, "A depth-first search algorithm for optimal PLA folding," *IEEE Trans. Computer-Aided Design*, vol. CAD-5, no. 3, pp. 433-442, July 1986.

- [Kang87a] S. M. Kang, "Metal-Metal Matrix (M^3) for high-speed VLSI layout," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, no. 5, pp. 886-891, September 1987.
- [Kern70a] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, pp. 291-307, February 1970.
- [Keut87a] K. Keutzer, K. Kolwicz, and M. Lega, "Impact of library size on the quality of automated synthesis," *IEEE Int. Conf. on Computer-Aided Design*, pp. 120-123, November 1987.
- [Koba89a] Y. Kobayashi, K. Asayama, M. Oohayashi, R. Hori, G. Kitsukawa, and K. Itoh, "Bipolar CMOS-merged technology for a high-speed 1 Mbit DRAM," *IEEE Trans. Electron Devices*, vol. 36, no. 4, pp. 706-711, April 1989.
- [Koll88a] P. Kollaritsch, S. Lusky, S. Prasad, and N. Potter, "CLAY: A malleable-cell multi-cell transistor matrix approach for CMOS layout synthesis," *IEEE Int. Conf. on Computer-Aided Design*, pp. 142-145, November 1988.
- [Lai87a] F. P. Lai, S. M. Kang, T. N. Trick, and V. Rao, "JADE: A hierarchical VLSI CMOS circuit optimizer," *Int. Conf. on Computer-Aided Design*, pp. 48-51, November 1987.
- [Leon86a] H. W. Leong, "A new algorithm for gate matrix layout," *Int. Conf. on Computer-Aided Design*, pp. 316-319, November 1986.
- [Lewa84a] J. L. Lewandowski and C. L. Liu, "A branch and bound algorithm for optimal PLA folding," *21st Design Automation Conf.*, pp. 426-433, 1984.
- [Lope80a] A. D. Lopez and H. F. Law, "A dense gate matrix layout method for MOS VLSI," *IEEE Trans. Electron Devices*, vol. ED-27, pp. 1671-1675, August 1980.
- [Mah84a] G. H. Mah and R. Newton, "PANDA: a PLA generator for multiply-folded PLAs," *Int. Conf. on Computer-Aided Design*, pp. 122-124, November 1984.
- [Mess89a] T. S. Messerges, "A novel switch-level logic simulator for VLSI MOS circuits," M.S. thesis, University of Illinois, Urbana-Champaign, June 1989.

- [Nage75a] L. W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," Electronics Research Laboratory, Memorandum No. ERL-M520, University of California, Berkeley, May 1975.
- [Naka80a] M. Nakaya, O. Tomisawa, I. Ohkura, and T. Nakano, "High-speed MOS gate array," *IEEE J. Solid-State Circuits*, vol. SC-15, no. 4, pp. 730-735, August 1980.
- [Noij85a] W. A. M. Van Noije and G. J. Declerck, "Advanced CMOS gate array architecture combining "gate isolation" and programmable routing channels," *IEEE J. Solid-State Circuits*, vol. SC-20, no. 2, pp. 469-480, April 1985.
- [Ohts79a] T. Ohtsuki, H. Mori, E. S. Kuh, T. Kashiwabara, and T. Fujisawa, "One-dimensional logic gate assignment and interval graphs," *IEEE Trans. Circuits and Syst.*, vol. CAS-26, no. 9, pp. 675-684, September 1979.
- [Ornd81a] R. M. Orndorff, M. M. Spanish, G. D. Gebhard, and D. M. Mitchell, "Advanced CMOS/SOS gate array technology," *Semi-Custom Integrated Circuit Technology Symp.*, pp. 221-240, May 1981.
- [Otte82a] R. H. J. M. Otten, "Layout Structures," *1st IEEE Large Scale Syst. Symp.*, 1982.
- [Oust84a] J. K. Ousterhout, G. T. Hamachi, R. N. Mayo, W. S. Scott, and G. S. Taylor, "Magic: A VLSI layout system," *21st Design Automation Conf.*, pp. 152-159, 1984.
- [Over89a] D. V. Overhauser, "Fast timing simulation of MOS VLSI circuits," Ph.D. dissertation, University of Illinois, Urbana-Champaign, October 1989.
- [Posl85a] S. D. Posluszny, "SLS: An advanced symbolic layout system for bipolar and FET design," *Int. Conf. on Computer-Aided Design*, pp. 346-348, November 1985.
- [Rao88a] V. Rao, D. Overhauser, I. Hajj, and T. Trick, *Switch-level Timing Simulation of MOS VLSI Circuits*, Kluwer Academic Publishers, Boston, 1988.
- [Reed85a] J. Reed, A. Sangiovanni-Vincentelli, and A. Santamauro, "A new symbolic channel router: YACR2," *IEEE Trans. Computer-Aided Design*, vol. CAD-4, pp. 208-219, July 1985.

- [Saka85a] K. Sakashita, M. Ueda, T. Arakawa, S. Asai, T. Fujimura, and I. Ohkura, "A 10K-Gate CMOS gate array based on a gate isolation structure," *IEEE J. Solid-State Circuits*, vol. SC-20, no. 1, pp. 413-417, February 1985.
- [Sech86a] C. Sechen and A. Sangiovanni-Vincentelli, "TimberWolf 3.2: A new standard cell placement and global routing package," *23rd Design Automation Conf.*, pp. 432-439, June 1986.
- [Smit82a] K. F. Smith, T. M. Carter, and C. E. Hunt, "Structured logic design of integrated circuits using the storage/logic array (SLA)," *IEEE J. Solid-State Circuits*, vol. SC-17, no. 2, pp. 395-406, April 1982.
- [Sout85a] J. R. Southard, "MacPitts: An approach to silicon compilation," *IEEE Computer*, pp. 74-82, April 1985.
- [Su87a] S. Su, V. B. Rao, and T. N. Trick, "HPE: A hierarchical parasitic circuit extractor," *24th Design Automation Conf.*, 1987.
- [Szab87a] K. S. B. Szabo, J. M. Leask, and M. I. Elmasry, "Symbolic layout for bipolar and MOS VLSI," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, no. 2, pp. 202-210, March 1987.
- [Tayl84a] G. S. Taylor and J. K. Ousterhout, "Magic's incremental design-rule checker," *21st Design Automation Conf.*, pp. 160-165, 1984.
- [Tera87a] M. Terai, Y. Ajioka, T. Noda, M. Ozaki, T. Umeki, and K. Sato, "Symbolic layout system: Application results and functional improvements," *IEEE Trans. Computer-aided Design*, vol. CAD-6, no. 3, pp. 346-354, May 1987.
- [Ushi88a] Y. Ushiku, T. Kobayashi, A. Yoshida, N. Itoh, A. Nishiyama, and R. Nakata, "An optimized 1.0-um CMOS technology for next-generation channelless gate arrays," *IEEE J. Solid-State Circuits*, vol. 23, no. 2, pp. 507-513, April 1988.
- [Wein67a] A. Weinberger, "Large scale integration of MOS complex logic: A layout method," *IEEE J. Solid-State Circuits*, pp. 182-190, December 1967.
- [West81a] N. H. Weste, "Mulga -- An interactive symbolic layout system for the design of integrated circuits," *Bell Syst. Tech. J.*, vol. 60, no. 6, pp. 823-857, July-August 1981.

- [Will78a] J. D. Williams, "STICKS -- A graphical compiler for high level LSI design," *National Computer Conf.*, pp. 289-295, May 1978.
- [Wime88a] S. Wimer, I. Koren, and I. Cederbaum, "Floorplans, planar graphs, and layouts," *IEEE Trans. Circuits and Syst.*, vol. 35, no. 3, pp. 267-278, March 1988.
- [Wing85a] O. Wing, S. Huang, and R. Wang, "Gate matrix layout," *IEEE Trans. Computer-Aided Design*, vol. CAD-4, no. 3, pp. 220-231, July 1985.
- [Wu85a] M. Y. Wu, W. Shu, and S. P. Chan, "A unified theory for MOS circuit design - switching network logic," *Int. J. Electron.*, vol. 58, no. 1, pp. 1-33, 1985.
- [Wu87a] M. Y. Wu and I. N. Hajj, "Modified sequential CMOS circuit design," *IEEE Int. Conf. on Computer-Aided Design*, pp. 460-463, November 1987.
- [Yang80a] P. Yang, I. N. Hajj, and T. N. Trick, "SLATE: A circuit simulation program with latency exploitation and node tearing," *Proc. Int. Conf. on Circuits and Computers*, pp. 353-355, October 1980.
- [Youn85a] J. Young, "IC-Design automation strides into silicon-compilation era," *Electronics*, pp. 58-63, June 24, 1985.
- [Yu85a] Q. Yu and O. Wing, "Interval-graph-based PLA folding," *Int. Symp. on Circuits and Syst.*, pp. 1463-1466, September 1985.